SANDIA REPORT

SAND2005-3356
Unlimited Release
Printed October 2005

Test, Evaluation, and Build Procedures For Sandia's ASCI Red (Janus) Teraflops Operating System

Daniel W. Barnette

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865)576-8401
Facsimile: (865)576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800)553-6847
Facsimile: (703)605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2005-3356
Unlimited Release
Printed October 2005

Test and Evaluation Procedures For Sandia's Teraflops Operating System (TOS) On Janus

Daniel W. Barnette
Scalable Systems Integration Department
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-1111

Abstract

This report describes the test and evaluation methods by which the Teraflops Operating System, or TOS, that resides on Sandia's massively-parallel computer Janus is verified for production release. Also discussed are methods used to build TOS before testing and evaluating, miscellaneous utility scripts, a sample test plan, and a proposed post-test method for quickly examining the large number of test results. The purpose of the report is threefold: 1) to provide a guide to T&E procedures, 2) to aid and guide others who will run T&E procedures on the new ASCI Red Storm machine, and 3) to document some of the history of evaluation and testing of TOS. This report is not intended to serve as an exhaustive manual for testers to conduct T&E procedures.

Acknowledgements

The author wishes to acknowledge the helpful discussions and insights given by Sandians Suzanne Kelly, John Van Dyke, Robert E. Benner, Gerald Quinlan (retired), Michael Hannah, and many others. More than once, the parallel system engineers group, or PSE's, with Frank Jaramillo as lead, helped with hardware and software problems. This effort was managed initially by Douglas Doerfler and subsequently by James A. Ang.

Table of Contents

Test and Evaluation Procedures for Sandia's Teraflops Operating System (TOS) on Janus

Introduction

Sandia National Laboratories' ASCI Red massively parallel computer, hostnamed Janus, is the world's first teraflops supercomputer. The machine was developed by joint effort of Intel and Sandia, built by Intel, and brought online in 1997 at Sandia in Albuquerque, NM. Although now dated and soon to be retired, Janus has been a mainstay for computing at Sandia for over eight years.

Teams from Intel initially managed the machine. Sandia personnel took sole responsibility for system management in January 2001 after several months of side-by-side training with Intel. Sandia will continue managing Janus until its decommissioning in FY2006.

As a part of Sandia's responsibilities for managing Janus, the Teraflops Operating System (TOS) Project was formed. The goals of the Project are to

- Periodically upgrade the system software with the goals of improving reliability and availability
- Provide computational support to the computer users
- Provide technical software consulting to the system administrators
- Respond to user questions on system software

This report focuses on a portion of the first goal: that of "test and evaluation" (T&E) methods used when upgrading the system software. Janus' operating system consists of TOS running on service nodes and the lightweight kernel called Cougar running on compute nodes. The purpose of T&E is to stress TOS/Cougar software and associated hardware to reasonably ensure that all parts of the system are operating and communicating as required. The goal of T&E is to discern whether regressions of TOS have occurred through changes and that its integrity and viability remain intact.

The importance of T&E cannot be overstated considering what is involved in a TOS/Cougar upgrade. To upgrade system software means TOS and/or Cougar changes will be made on a continuing basis. Additions, updates, and/or improvements, whether for purposes of simplifying, adding capability to, or debugging code, are assumed to render TOS/Cougar untrustworthy until subjected to rigorous T&E procedures. Also, a massively-parallel computer's operating system is designed to have many processors communicating with each other as well as with parallel file systems and other peripheral devices, all with multiple concurrent users. With the OS so complex, and with Janus supporting a large user base solving problems of national interest, the ability to verify system integrity and reliability is crucial. Hence, T&E procedures must be performed after each time the operating system is changed and before the start of production runs.

Since Sandia took over sole management of Janus, over 24 TOS/Cougar releases have been tested. The vast majority of these passed T&E and were approved for production status. However, T&E uncovered problems with several releases. The problems were either corrected or the proposed fixes were scrapped. Many times problems that arose were not related to the proposed code fixes but to interaction of the proposed fixes with other sections of TOS/Cougar. Only intensive T&E could hope to uncover such problems.

In particular, this report describes the T&E methods by which Sandia's TOS/Cougar is evaluated before production release. Janus hardware and software are discussed first. Next, support computers for T&E are detailed. Steps for building TOS/Cougar are given next, and then an overview of testing TOS is discussed. Finally, a method for post-test analyses is proposed.

Janus Teraflops Computer and Its OS

Janus is a distributed memory, MIMD, message-passing supercomputer. I/O, memory, compute nodes, and communication are highly scalable. The system uses two operating systems: Unix-based TOS for the user interface and Sandia-developed Cougar for non-intrusive scalable application on the compute nodes. Affordability is maintained through use of commercial-off-the-shelf (COTS) technology where possible. Details can be found in Ref. 1.

Janus' mesh topology consists of 38 X 32 X 2 = 2432 card slots, or room for a maximum of 4864 physical nodes, there being two nodes on each card. Each node contains two processors, one for computing and another for networking. The networking processor can be configured as a compute processor in what is known as a virtual compute node. Hence, each card contains two processors in normal mode or four processors in virtual mode.

There are 4562 (4510 compute + 52 service) of what are known as kestrel, or compute, nodes. There are 2 physical nodes per kestrel board, accounting for 4562/2 = 2281 cards, leaving 2432 - 2281 = 151 "other" card slots.

I/O, network and boot nodes are implemented on eagle boards. There are 73 I/O + 2 boot + 12 network = 87 of these nodes, but each node uses a whole card slot. So, 151 - 87 = 64 empty card slots.

In summary, there are a total of 4510 physical or 9020 virtual compute nodes when Janus is in the normal configuration. This can be augmented by reconfiguring other nodes to be compute nodes if necessary, but this is seldom if ever done. Obviously, though, there must always be some minimum number of service and I/O nodes available at all times.

Service nodes running TOS and compute nodes running Cougar are implemented on kestrel boards, with two nodes of 2 processors each, 4 processors total, per board. There are a total of 52 service nodes running TOS.

The system's resource allocations are flexible. For example, service nodes may be used as compute nodes as long as the proper number of boot, I/O, and network nodes remains available. This would raise the total number of compute nodes and decrease the number of service nodes. Of course, a normal service node to be used in the compute partition must be instructed to boot the Cougar OS instead of TOS.

Janus uses a Parallel File System (PFS) designed to provide high I/O bandwidth required for parallel applications. The PFS file system is compatible with other UNIX file system types such as UFS and NFS, and can be mounted in the system-wide directory hierarchy in a similar fashion. A PFS file is striped in a round-robin fashion across a group of regular Unix File Systems (UFS). Each UFS file system in the stripe group may be created on a different storage device. These storage devices can be connected to one or several distinct I/O nodes. Multiple PFS file systems may be mounted in the system at a time, each with different default data striping attributes and buffering strategies.

Janus is typically divided into classified and unclassified sections. The unclassified section is further divided into an interactive partition, where small jobs can be submitted directly by the user, and a queued partition, where large jobs are submitted to a queuing algorithm. T&E is always run on the unclassified side of Janus. No testing occurs on the classified side. Any discussion related to the classified side of Janus is beyond the scope of this report.

As Sandia personnel gained experience with Janus, it was noticed that the support group became inundated with emails when Janus incurred problems. It was decided that so many emails might be more easily managed if divided into groups, or lists, rather than in bulk. This allowed support staff to better focus on problem areas. As a result, four email lists were created for communications between support staff, and between staff and users. The lists and their purposes are given below.

janus-help:	this is the list that users should use and the only list to be recommended by support staff to users. When a user mailing is answered, support should always CC this list at least so the rest know that it has been answered, to avoid duplicate replies to the user. The resolution of a query should also be sent to this list so that all the others can learn from the answer. It may be appropriate to have one-on-one discussions with the user as part of diagnosing the problem, but the final resolution should always be sent to this list. Mail to this list is automatically archived.
janus-admin:	this is only for concerns about the e-mail lists themselves, and is where requests to be added or deleted from a janus list are sent. All janus mail "bounces" go to this list.
janus-sys:	this is for the internal use of the systems and support staff as they discuss issues amongst themselves. It should not be given to users as an address for them to by-pass janus-help
janus-sw:	this is for the internal use of the software staff as they discuss issues amongst themselves. It should not be given to users as an address for them to by-pass janus-help

Building TOS for Testing and Evaluation

After developers finish making changes for a proposed release, the updated TOS is checked in to the version control software systems CVS (Ref. 2). The tester then checks out this newest TOS version from CVS and initiates the build process. A complete build requires approximately a 24-hr day if all goes correctly.

The following steps are used during a typical build process for TOS. The build steps must be followed carefully. If any mishaps or mistakes occur, the build process is typically started over from the beginning. There is no restart process available during the build.

Step 1. Tests are checked out of the CVS repository into a local directory called ‘sandbox’. (30 minutes)

The four most-useful CVS command sequences for importing files, exporting files, test check-outs, and test check-ins are listed below. Note that the double backslashes “\\” indicate line continuation.

1. Import files into CVS repository:

```
cvs -d repository  import  -m "Import Source Code"    \\
                    CVS_module_name   xx   start
```

2. Export files out of CVS repository:

```
cvs   -d repository   export   -D today   CVS_module_name
```

3. Check-out files from CVS for immediate use:

```
cvs   -d repository   checkout   -d checkout_directory_name  \\
        CVS_module_name
```

which will put files that were in CVS_module_name into “checkout_directory_name”, or

```
cvs -d repository checkout CVS_module_name
```

which will put files that were in CVS_module_name and its directories into the current directory.

4. Check-in of a modified file that already resides on CVS:
First, “cd” to the directory which contains the modified file. Then issue the following three commands.

```
cvs update

cvs add <filename>

cvs commit <filename>
```

Step 2. All files in the ‘sandbox’ directory are gathered together using the ‘tar’ utility to a file nominally called ‘source.tar’. (15 minutes)

Step 3. The ‘source.tar’ file is secure-copied to the build machine. (1.5 hours)

Step 4. The ‘source.tar’ file is untarred at this point. (2.25 hours)

Step 5. Appropriate links are made, extraneous build files are moved into place, and environment variables are set using ‘setenv’ and sourcing a setup script. (15 minutes)

Step 6. The executables are built using ‘gmake’. (14 hours)

A continuous connection is required between user and build machine during the build. This is trivial if the user is onsite at Sandia. If offsite, the user may use open-source Virtual Network Computing (VNC) software (Ref. 3). Details for using VNC in the context of a TOS build are given in Appendix A.

Step 7. The build script creates a ‘Release’ directory.

Step 8. Files in the ‘Release’ directory are gathered using the ‘tar’ utility into a file nominally called ‘build.tar’.

Step 9. The 'build.tar' file is moved to a directory where the Parallel System Engineers, or PSE's, can access it. The PSE's then install the newly-built TOS on the evaluation machines for testing and, if the tests pass, on Janus for final testing. If it passes the final test, this TOS version becomes the resident operating system for production runs. Testing TOS will be discussed in the next section.

Step 10. Anytime after successful testing but before another TOS version is checked into CVS, the build files are stored on the Sandia Mass Storage System, or SMSS (Ref. 4).

Step 11. Finally, the CVS files are tagged with the current TOS version number for tracking purposes using the command

```
cvs tag <version number> sandbox
```

Testing and Evaluating TOS – An Overview

Attention is now turned toward testing and evaluating the newly built, but not yet tested, TOS. For this purpose, Intel provided several mini-teraflop machines to Sandia as a contract requirement, with sizes varying from tens to hundreds (versus Janus' thousands) of processors. This allows Janus to be essentially dedicated to users while the mini-teraflop machines can be used for various support functions. With a total of 14 support machines, Sandia uses four almost exclusively for testing and evaluation. These are hostnamed Basil, Rosemary, Ginger, and Nighten. A complete list of support machines and their purpose is given in Appendix C.

It was previously noted that the T&E tests were taken over by Sandia staff from Intel personnel in December 2000. Intel staff was minimal by this time, and little attention was being given to the numerous coding errors in the test scripts. Once Sandia staff took over in CY2001, high priority was given to correcting the errors. The most obvious or important ones were fixed first. Others were found during subsequent testing and were corrected. A comprehensive list of Sandia-initiated code fixes is given in Appendix D.

A. The ddt Test Harness

For most of the tests, the Driver Dependency automated Test execution tool (ddt) test harness assists in automated execution by using the test directory structure, user environment, and user specified options. The driver traverses the specified test directories to lock a test directory, then compiles and executes the test to generate the specified results. The driver attempts to successfully execute each test. The defaults are to execute a maximum of 5 circuits of the test directory file using the default master "Makefile". User-specified deviations to the defaults are allowed.

ddt parses the command line arguments until an unrecognized argument is found. This first unrecognized argument, and any remaining arguments, is passed directly through to "make". In this way options and macros may be specified and passed to "make" easily. This implies that ddt cannot share options with the "make" utility.

Any optional targets for "make" may also be specified, i.e. build, logs, etc. A "make" target is the name of the dependency target to be made. The default "make" target is "all", the UNIX custom.

The evaluation environment includes variables required for the ddt shell to execute properly. Environment variables describe the location of the ddt directory structure, host and mesh configuration. Most are set by ddt when it executes, or they can be permanently placed in the user's environment.

Key variables set by the ddt shell are shown in Appendix E.

B. Steps for Testing and Evaluation of TOS

T&E tests are stored in a CVS repository in much the same manner as TOS. CVS storage allows part or all of the testing software to be retrieved or checked out, changes to the tests to be tracked, and changes to be checked in.

Once checked out of CVS, the T&E suites are used to check system calls, commands, job queuing, libraries, communications, file systems, and other functions of TOS. Each test generates pass/fail output. A summary for groups of test runs is typically generated at the end of the test. The summary indicates a pass/fail for each group, allowing the tester to easily find individual test failures once the group is known.

Detailed instructions for running the evaluation tests are located on a password protected website on the Sandia Restricted Network. Contact the author if access is needed. Sample web pages from the EATS test are shown in Appendix F. All other tests have similar web pages.

The following steps explain typical test procedures.

Step 1: Eval tests are checked out of CVS using Script 1 in Appendix G. The script is abbreviated for brevity. All tests are placed in proper directories for running on the mini-teraflop machines.

Step 2. The EATS (cf. Appendix C, Test 1, and Appendix F) is always run first. This is a general test that uses parts from other tests to check TOS. Several groups of tests are run. Each group runs relatively quickly compared to the later, more extensive tests. An EATS pass provides an indicator of basic system functionality and whether obvious regressions might have occurred. If this test fails and the failure cannot be attributed to anything other than the revised OS, no other tests are run and the OS is scrapped.

Step 3. Most other tests must run on specific machines. These tests are launched simultaneously on the four support machines. As one test finishes and results are analyzed, another is begun. Some tests take over 24 hours to finish.

Step 4. The tests are monitored regularly for failures. A test failure is always investigated as to cause. If code related, the error is fixed and the test re-run. If it can be ascertained that the test failure is TOS related, this version of TOS is scrapped.

Step 5. There are two tests that are run only on Nighten: Munops and IO-Munops. These tests are typically saved until near the end of T&E. If the TOS changes are I/O related, IO-Munops is run; if not, Munops is run. The tests simulate concurrent users constantly submitting small jobs of 5-10 minutes run time. This stresses the queues, job allocater, mesh partitioning, and TOS, among others. These tests require a run of 24 hours each on Nighten without a TOS- or Cougar-related crash or hang. Before each Munops or IO-Munops run, a system check is run and recorded using Script 2 in Appendix G, “System Status Checker.”

Step 6. Finally, Munops or IO-Munops is run on Janus during dedicated system time. More details on this step are given in Section F.

C. Running mini-Eats in the Janus Interactive Partition

Only one seldom-used test was designed to run in Janus’ interactive partition during normal production runs. The mini-EATS tests are scaled-down versions of the EATS tests. Some of the tests require a minimum number of processors, but all can run in the interactive partition’s default size. The idea, then, is to run at least some T&E tests “on the fly” without taking the entire system for dedicated testing.

However, attempting to run mini-EATS in the interactive partition while other users are on the machine can lead to conflicts such as unavailable nodes or not enough processors for a particular test.

A script was written so that each mini-Eats test continually checks the size of the partition for the current nodes available. The script is given as Script 3 in Appendix G, "Pinging Interactive Partition Until Specified Number of Nodes Becomes Available." If the interactive partition is full and no nodes are currently available, the script keeps trying to allocate nodes for a user-specified number of tries, each a user-specified number of seconds apart. This permits the main script to wait for nodes to become available to automatically continue T&E without user intervention. If nodes do not become available within the specified time, the script gives up and quits. In this case, it is assumed something is wrong and is outside the ability of the script to fix it. Once available, however, the proper number of nodes is allocated immediately before each mini-Eats test is run. This approach usually allows the tests to continue even though the partition sees heavy use. The script can easily be modified for use with any interactive job.

D. Running Specific Tests for Specific Code Fixes

Certain T&E tests may be run for TOS code fixes targeted to specific functionality. This allows the tester to avoid running irrelevant tests and thus significantly reduce time to results. The TOS portions affected by revision and the corresponding test suite to run are listed below.

Portion of TOS affected by Fix	**Test Suites To Be Run**
NQS (network queuing system)	EATS, NQS
COUGAR (compute node OS)	COUGAR EATS, Parallel Apps, MUNOPS
TOS (server/kernel, on service nodes)	Essential: EATS, AutoUnix, MUNOPS Possible other tests include: IO-Munops
PFS (parallel file system)	EATS, File I/O, MUNOPS

E. In the Event of a Test Failure

If a test fails, the test is usually re-run. If a test subsequently passes on re-run without any changes, the pass may be due to a myriad of reasons. Communication links, network problems, and conflicts with other tests running simultaneously have all been known to cause test failure. In these instances, simply re-running the test may result in success.

If the test failure is due to code error, then the error is fixed and the revised code checked back in to CVS. A comment as to what the error was and how it was fixed is usually included in the check-in process. The test is then re-run.

If a test hangs or crashes TOS, the following four steps are taken:

> Step 1. Before the machine is rebooted, crash data is collected by a script called **STAMPEDE**. **STAMPEDE** resides on the SPS workstation for the relevant machine.
>
> Step 2. The latest **PEEKABOO** process output is located. **PEEKABOO** is a **cron** script that runs every 5 minutes. This job issues commands such as **ps**, **showmesh**, and **stat** to check the responsiveness of the system and write the data to a directory.

Step 3. **STAMPEDE** data is to be **ftp**'d to a computer called **Sylvester** which resides in Bldg. 980 as of Feb, 2001. The **PEEKABOO** process data is usually attached to the **STAMPEDE** data before **ftp**'ing.

Step 4. The software engineers examine the data and try to decide what caused the evaluation system to crash. If the crash is relevant to TOS and has not occurred in previous tests, the problem is isolated, the change related to the problem is fixed, the TOS version number is scrapped, and work begins on incorporating the revisions into the next TOS release.

F. Final Test on Janus

By this time all of the T&E tests have been run on the relevant support machines. If all tests receive a pass, a final test is run on Janus during dedicated system time. The only test to be run at this point is either Munops or IO-Munops (cf. Appendix B, Tests 29 and 30). The test is run for approximately six hours.

Before running the final Munops or IO-Munops test on Janus, the tester compiles a test plan detailing steps that will be taken during the test. The test plan is emailed to all personnel involved in the test. A typical test plan for the version R4.5.2 TOS release is given in Appendix H.

A typical Munops/IO-Munops test failure results in jobs not completing or a machine crash. Then, Janus is examined from the console for clues of either TOS failure or un-related problems. If problems are traced back to the proposed version of TOS, the test is considered as failed. This results in the proposed TOS being scrapped, even if all of the previous tests on the support machines have passed. If the problems are traced to something other than TOS, the machine is rebooted and the test continues. The decision for GO or NO-GO production installation for the proposed TOS is made by a minimum of three personnel near the end of the test run.

If this and all previous tests have been successful, the revised TOS is scheduled to be installed on Janus at a later date, after users have been notified of the impending TOS change.

Proposed Methods for Post-Test Analyses

Some T&E tests consist of numerous sub-tests. For example, the MPI test suite runs over a thousand sub-tests. Manually looking for pass/fail criteria, as well as discrepancies or regressions, for failed sub-tests can be a daunting task. It becomes desirable, then, to automate this task such that failed tests become readily obvious. Described herein is such a method.

In Appendix I is given a script to gather results from the different directories in which T&E tests are run. The user is expected to ensure that the test results always wind up in the specified directories and files whether the test succeeds or fails.

Once they are gathered, the results are run through a Mathematica (Ref. 5) script that makes use of Mathematica's rich graphics sets. This script is given in Appendix K. Shown are the first 25 MPI test results gathered and plotted using Mathematica. Test run time is plotted versus test number. Bar graphs are used to show test results. If a test fails, the bar for that test only is colored red and the run time is given a negative value. As shown, this causes the failed test to stand out readily. Also, test times can be compared visually with previous tests to make sure run times are as expected. If these run times are significantly different, a test regression might be implied.

Finally, the script prints out a list of tests. The tests are numbered to correspond to the numbers indicated on the bar graph's abscissa. Although only bar graphs are shown in Appendix J, several other plot types, such as pie graphs, were investigated for their usefulness in presenting test results. The information gleaned from these was surprisingly deficient as a means to readily draw conclusions from test results. At least from the plot types experimented with by the author, the bar graphs were the most useful by far.

References

1. "ASCI Red – The World's First TeraOps SuperComputer," Sandia National Laboratories, Albuquerque, New Mexico. http://www.sandia.gov/ASCI/Red/index.html

2. "CVS Concurrent Versions System: The open standard for version control." https://www.cvshome.org

3. "About RealVNC.." http://www.realvnc.com/

4. "Servers – SMSS Open," Sandia National Laboratories, Albuquerque, New Mexico. http://www-irn.sandia.gov/analyst/server/smss-open.html

5. "Mathematica – The Way the World Calculates," Wolfram Research Incorporated, Champaign, Illinois. http://www.wolfram.com/products/mathematica/index.html

Appendix A. Using VNC for Maintaining Connectivity from Remote Sites

What you'll need:

- A computer running VNC (Virtual Network Computing) server. Machine happens to be rs2l2.sandia.gov (that's rs2"ell"2).
- An ssh client running on your local machine; Sandia uses F-Secure's ssh client
- A VNC client (called TightVNC) running on your local computer. This is a free download from

http://www.tightvnc.com/download.html

Only get the viewer, if possible, since you will NOT need the server program.

- It helps to read "Getting Started with VNC" at

http://www.realvnc.com/gettingstarted.html

or refer to the following for more info. SOME OF THE DESCRIPTIONS IN THESE WEBSITES ARE CONFUSING AND MISLEADING -- USE WITH CAUTION.

http://www.uk.research.att.com/archive/vnc/sshvnc.html
http://www.realvnc.com/
http://www.cs.hmc.edu/tech_docs/qref/vnc.html#ssh_tunnel

The last one gives a nice description of tunneling in the section "Basic SSH Tunneling" and describes the "5900" and "5901" as used below.

Getting Connected

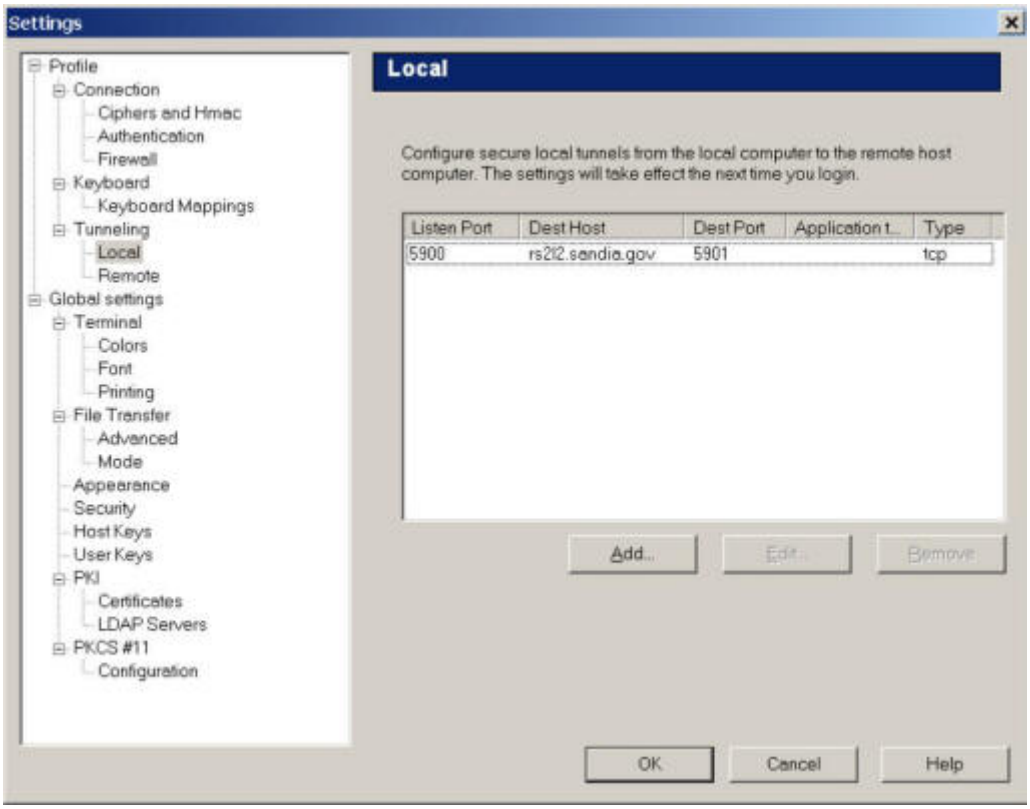
1. **Set up your F-Secure ssh client to tunnel**

Open the ssh client on your local machine. Go to the "services" icon on the ssh taskbar. Click on "Tunneling" in the left sidebar and check the box "Tunnel X11 Connections". Leave the "X11 Display" set to zero.

Next click on "Local" under the "Tunneling". Click on "Add" and enter the following data:

```
Listen Port (aka Source Port): 5900
Destination Host: rs2l2.sandia.gov
Dest Port: 5901
Application to start: (leave blank)
Type: tcp
```

When you are finished, the window will look like this:



The last two digits of 5900 imply that the VNC server will send its data back to display :0 on your local machine (valid for Windows). The last two of 5901 imply that we will set the VNC server on the remote machine to display on :1, as will be shown below.

Close the above window by clicking on "OK".

2. Login to rs2l2.sandia.gov

Use your SSH window to login to rs2l2.sandia.gov.

3. Create a VNC server password just for you

To create your VNC password, type

```
rs2l2% vncpasswd
Password:
Verify:
rs2l2%
```

As shown above, you can enter a Password, then you are asked to Verify. You will need this password in Step 5's "Session Password" in the VNC Authentication box.

4. Start the VNC Server on rs2l2.sandia.gov

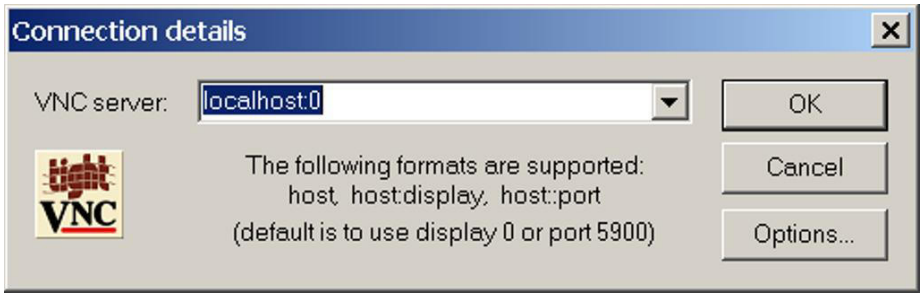
Start the VNC server as follows:

```
rs2l2% vncserver :1

New 'X' desktop is rs2l2:1

Starting applications specified in /home/dwbarne/.vnc/xstartup
Log file is /home/dwbarne/.vnc/rs2l2:1.log

rs2l2%
```

The :1 will be understood now as port 5901. If you had used 5906, then you would need to enter "vncserver :6".

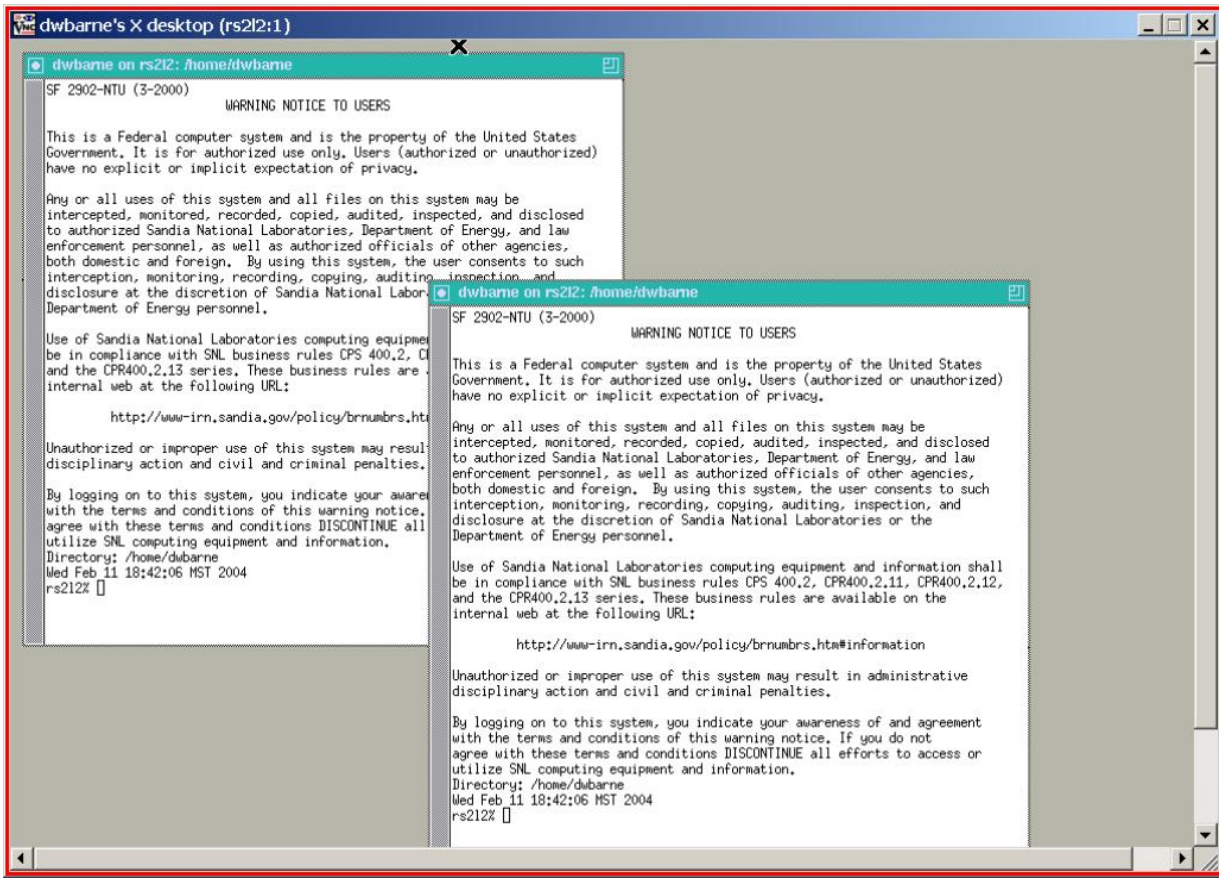
5. Start your local vnc client

Start your local vnc client (click on the icon) that you installed earlier. In the box provided, enter "localhost:0" (no spaces) as shown below (provided you entered "5900" above in Step 1; if you entered "5905" instead, for example, then you would enter below "localhost:5"):



Next, enter your session password that you created in Step 3.

An X-window will now pop up on your local display with at least one xterm window logged in to rs2l2:



From here, use the xterm windows to login to another machine and launch a process.

You can then close the above window (do NOT logout of the xterm windows!! Just click on the "X" in the top far right corner of the larger window!!) and the process will continue to run, even if you shut down your local machine.

The VNC server keeps the connection open between the xterm window and the other machine you logged in to from the x-term window, unless one of the machines stops running for some reason.

In fact, you can completely shut down your local machine, start it any time later, repeat the login process, and find the xterm window as if you never logged out (provided both machines continued to run).

6. When you are finished...
It is a good idea to stop your VNC server on rs2l2 when you are finished. Do this by typing

```
rs2l2% vncserver -kill :1
Killing Xvnc process ID 20045
rs2l2%
```

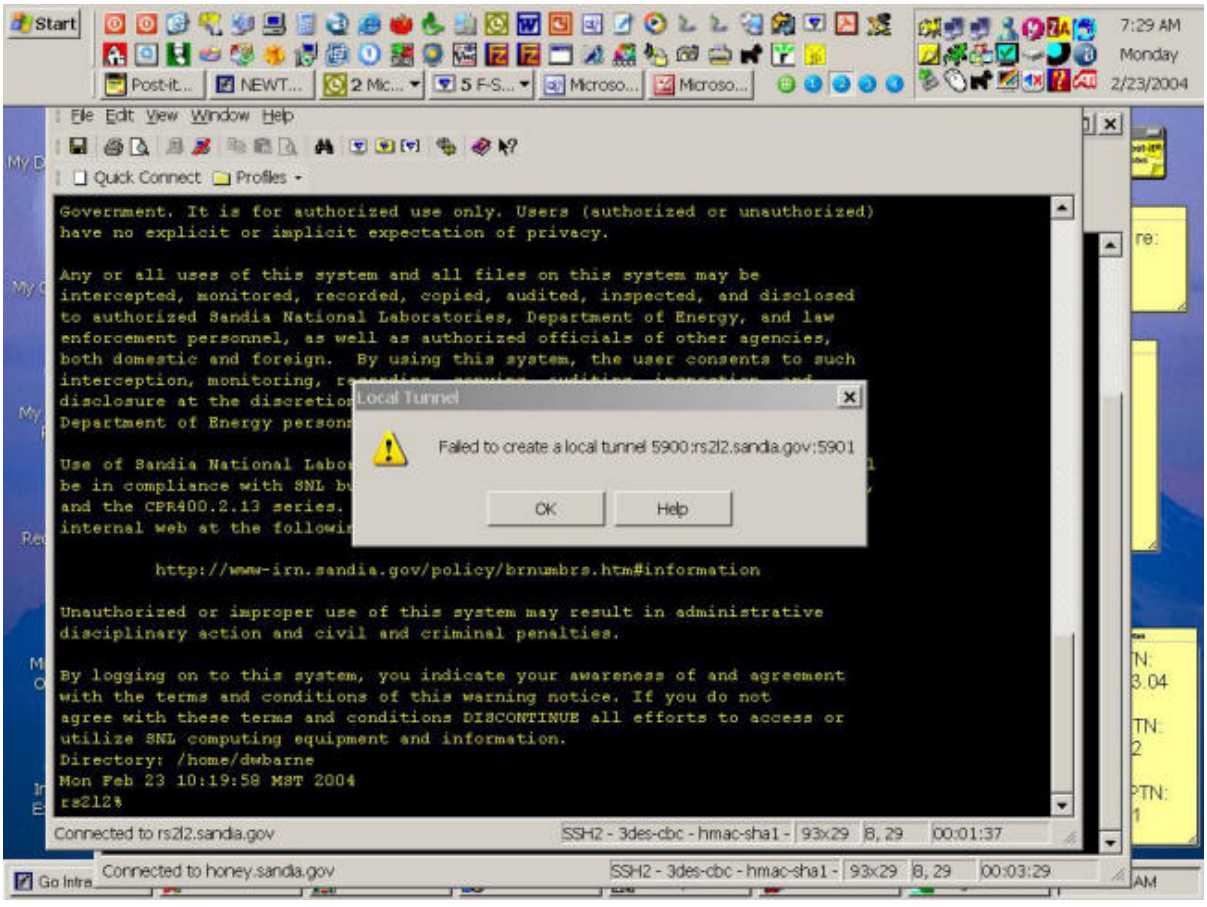
Another way to stop VNC servers you have started is to kill the process ID:

```
rs2l2% ps -ae | grep vnc
20360 pts/0 00:00:00 Xvnc
20436 pts/0 00:00:00 Xvnc
20510 pts/0 00:00:00 Xvnc
rs2l2% kill 20360 20436 20510
rs2l2%
```

Problems

The tunnel cannot be created

Several times I have run into the problem that the tunnel cannot be created. This can happen if Listening Port :0, or whatever Listening Port you were using, is in use for some reason (you may not know the reason!). The problem happens immediately when trying to login to rs2l2 using ssh. The pop-up window is



If this happens, you will not be able to start a VNC connection.

The solution appears to be that the Listening Port number needs to be changed. Hence, repeat the procedure shown in Step 1, but this time enter a Listening Port number of 5905, for example, instead of 5900. You will not need to change the "vncserver" number on the remote computer. Now, login to the remote computer. Then, in Step 5, instead of entering "localhost:0" in the VNC box, enter "localhost:5" (no spaces). A tunnel connection should begin.

IF YOU DECIDE THAT STOPPING THE VNC SERVER WILL HELP THIS PROBLEM, BE ADVISED THAT THIS WILL ALSO RE-START YOUR X-TERM WINDOWS ON THE REMOTE MACHINE, AND ALL OF YOUR LAUNCHED PROCESSES WILL BE STOPPED! THIS IS NOT A DESIRABLE SITUATION, SO USE THIS WITH CAUTION!

Appendix B: List of Support Hardware

Name	Service (TOS) Nodes	Compute (Cougar) Nodes	Disks	Purpose
1. basil	6	20	1 2GB bootdisk 1 RM20 disk (LUN 0 on io1, LUN 1 on io1/ch1)	Run eval tests for upgraded TOS
2. ebony	8	16	1 2GB bootdisk 1 RM20 disk (LUN 0 on io1, LUN 1 on io1/ch1)	Non-specific software testing
3. ellymay	3	8	1 2GB bootdisk 1 2GB scratch disk	Compiler development
4. felix	4	26	1 4GB bootdisk 1 4GB /home/projects/eval/share disk 1 2GB /home/projects/eval/users disk 1 4GB /home/projects/eval/users/dwbarne disk 1 2GB /home/projects/eval/users/rebenne disk	Cougar (compute node) software development
5. gale	0	0	Can be combined with nighten (becomes nightengale) but nodes are usually allocated to nighten in jumbo mode, which is why no nodes are listed at left	Non-specific software testing; shared resource
6. ginger	6	12	1 2GB bootdisk (w/custom bootdisk label) 2 RM20s pfs scratch disks	Run eval tests for upgraded TOS; ATM, ethernet, TOS, NFS, TCP/IP, and file I/O testing
7. granny	3	10	1 2GB bootdisk	Development system used by Parallel System Engineers, or PSE's)
8. honey	3	8	1 2GB bootdisk	TOS builds
9. jethro	3	10	1 2GB bootdisk	Run eval tests for upgraded TOS; non-specific software testing
10. jrflop	5	16	1 4GB bootdisk	Cougar (compute node) software development
11. nighten	30	144	1 2GB bootdisk 1 2GB /home/projects disk 6 RM20s scrach/pfs disks	Run eval tests for upgraded TOS
12. polaris	3	24	1 2GB bootdisk	T-FLOPS Board Repair Node Burn-in System
13. rosemary	4	8	1 2GB bootdisk 1 2GB /home/projects disk	Run eval tests for upgraded TOS
14. thyme	6	4	1 2GB bootdisk	Run eval tests for upgraded TOS (seldom used)

Appendix C: List of Evaluation Tests

No.	Name	Focus	Source	Comments
1.	EATS (Evaluation Acceptance Test Suite)	Always the first test run; general sanity test	Bits and pieces from many of the other tests	basic system functionality and Cougar kernel
2.	VSX R3.205 X/Open System Validation Suite	UNIX / POSIX	www.opengroup.org	Not used. Cost $20K for a ten-year license for t-flops. Is now available as test suite for the POSIX.1 aspects of the Linux Standard Base.
3.	SVVS (1985 System V Verification Suite) Standard Test	UNIX / System V	No current owner; possibly SCO	Not used.
4.	VSE (1991) OSF/1 Validation Suite Extensions 1.01	UNIX / OSF/1	Our documentation is from the Open Software Foundation;www.osf.org	Not used.
5.	VSX PFS (Parallel File System) Test	UNIX/POSIX with IO going to pfs file system	See above	Not used.
6.	SVSS PFS Test	UNIX/System V with IO going to pfs file system	See above	Not used.
7.	VSE PFS Test	UNIX/OSF/1 with IO going to pfs file system	See above	Not used.
8.	AutoUnix	TOS /sbin and /usr/bin programs	Intel	Developed at Sandia.
9.	DDT File I/O	UFS, NFS, and PFS	Intel	usability of Unix & NFS-mounted file systems; usability & performance of parallel file system DDT is the name of test harness
10.	Other/Non-DDT File I/O	UFS, NFS, and PFS	Intel	usability of Unix & NFS-mounted file systems; usability & performance of parallel file system
11.	DDT Sockets Test	Networking commands	Intel	Tests ifconfig, netstat, route, as well as socket functions
12.	NFS Test	NFS for ATM/Ethernet	www.caldera.com	The lachman test
13.	TCP-IP Test	TCP-IP for ATM/Ethernet	www.caldera.com	The lachman test
14.	RAID Utilities Test	Disk system administration	Intel	Developed at Sandia for LSI RAIDS Not used.

15.	Tfallocator and showmesh Test	Node allocator	Intel	Tests the node allocator, the display utility showmesh, as well as the mkpart (make partition), rmpart, and lspart
16.	Cougar yod/fyod Test	Tests all yod command line options.	Intel	job launch & parallel file i/o; Exercises a large part of the Cougar compute node OS; tests single app, shared app, and heterogeneous app. Runs through all fyod options including –munix and –masync IO options.
17.	Math Libraries Test	blas, pblas, lapack, libwc, openmp and fft libraries	No Intel copyright, although they are adapted to ddt test harness. Pblas references scalapack@cs.utk.edu	Tests BLAS, LINPACK, and any other scientific math libraries.
18.	NQS Test	Network queueing system (batch job submission process)	Intel	Batch system options.
19.	MPI Test	MPI-1 function correctness	Public Domain	Variation of the MPI test suite from MPI forum
20.	AutoDebug Test	Tests debug and xdebug	Intel	A very exhaustive test of every debugger command
21.	Parallel Apps/Message Passing Test	Tests the old Intel NX message passing library	Intel	Intel specific (NX library)
22.	RAS Test	Resiliency	Intel	"RAS" is the T-Flops Reliability, Availability, Serviceability component of the system. It involves agents on both the T-Flops system and SPS station. One of the most important agents is the T-Flops daemon rasd, as well as the tfallocator, bootmesh daemon, nqs daemons and the SPS services. This test suite is a component level suite, testing the interactions between the SPS station RAS services and the T-Flops daemons.
23.	General Regression Test	Verifies fixes to resolved problem reports are still working.	Intel	Tflops specific; This is actually a very handy suite to have in place. It contains the unit tests for resolved problem reports. It is a nice check that you haven't lost source code changes and/or that a different change somehow broke an OS fix.
24.	Pmake Test	Make, Pmake	O'Reilly, IBM, Intel, VSE	Intel certainly exercised make. There are four different tests of 'make'. One is from the O'Reilly book, "Managing Projects with Make". Another is an IBM POSIX conformance test. The raya tests were written by Ray Anderson at Intel. The last one seems to be part of the VSE test.
25.	Performance Monitor (AutoTprof) Test	Profiler tools	Intel	Intel wrote their own profile tools, and therefore also the test suite.

26.	AIM benchmark	TOS Capacity stress	No current owner; Can probably use for Red Storm	Not used. Copyright says AIM Technology Inc., but can find no record of this company. Instructions say: The test is done when system hangs or crashes or finishes. If it finishes, try rerunning with a larger end # of users. The more service nodes used, the more likely the end # will need to be larger since larger service partitions can handle a higher peak load. Typical results are for the test to make it to 60+ users and to terminate with CR 011419.
27.	SDET Benchmark	TOS Capacity stress System level benchmark designed to measure the system performance of any computer system.	Standard Performance Evaluation Corporation (SPEC)	Instructions say: The test is done when system hangs or crashes or finishes. If it finishes, try rerunning with a larger nn. A typical run will make it to 25+ users. If the test panics, hangs or terminates before 25 users, it may indicate a regression. Stresses a variety of system components (e.g. CPU, I/O, memory, operating system and many UNIX utilities). The methodology measures performance by systematically increasing the workload on the system. The system throughput increases with the workload until some system component (e.g. CPU, memory, I/O) becomes a bottleneck.
28.	KENBUS benchmark	TOS Capacity stress	Standard Performance Evaluation Corporation (SPEC)	All of the above comments for SDET apply here. For KENBUS the single performance metric is "scripts per hour"' and the maximum value is reported, with unconstrained freedom to vary the concurrency level. Simulates interactive users at keyboards
29.	MUNOPS (**M**ulti-**U**ser **NO**noverla**P**ping **S**tress) Tests	can the system take a user load?	Intel	Simulates user load observed causing problems in the wild. Fills the mesh with various compute-intensive and MPI-intensive tests (about 5-10 concurrently). Service partition is kept busy with mathlib tests. Each test runs 5-30 minutes, so lots of stops and starts.
30.	IO-Munops	Can the system take a user load with lots of IO?	Intel	Same as Munops above, but adds IO-intensive operations to the mix

Appendix D: List of Evaluation Test Suite Code Fixes

No.	Date bug fix incorporated into test suite [mm-dd-year]	1. Test Suite 2. Location of Tar File	Directory of file after test suite is untarred	Affected File and Comments
1.	12-21-2000	1. TFLOPS NQS 2. /home/SANDIA_EVAL/TESTS/NQS/sandia_nqs.tar.Z	<untar_dir>/Auto/libc/lib <untar_dir>/Auto/libf	Added two Auto subdirectories (libc/lib and libf) to the tar file since NQS uses them
2.	12-21-2000	1. TFLOPS NQS 2. /home/SANDIA_EVAL/TESTS/NQS/sandia_nqs.tar.Z	<untar_dir>/TF_NQS/base/qsub/qsub.sh	Changed backup and restore copies of sched_param to use cp -p; rather than cp
3.	12-21-2000	1. TFLOPS NQS 2. /home/SANDIA_EVAL/TESTS/NQS/sandia_nqs.tar.Z	<untar_dir>/TF_NQS/regressions/interactive_035626/interactive_035626.sh	Changed backup and restore copies of sched_param to use cp -p; rather than cp
4.	12-21-2000	1. TFLOPS tfallocator and showmesh 2. /home/SANDIA_EVAL/TESTS/ALLOC/sandia_alloc.tar.Z	<untar_dir>/lib	Changed function verify_lspart_output() in file 'yod_functs.ksh' to test only the first 8 characters of owner and group
5.	05-11-2001	1. IO-Intensive Munops 2. /home/SANDIA_EVAL/TESTS/MUNOPS/io_munops.tar	/home/projects/eval/io_munops/compute_sat/multi_rw	In file 'Makefile', the statement rm -f $(OBJS) mulit-rw was changed to rm -f $(OBJS) multi-rw to correct the spelling of 'mulit'
6.	05-11-2001	1. IO-Intensive Munops 2. /home/SANDIA_EVAL/TESTS/MUNOPS/io_munops_for_janus.tar	/home/projects/eval/io_munops/compute_sat/multi_rw	In file 'Makefile', the statement rm -f $(OBJS) mulit-rw was changed to rm -f $(OBJS) multi-rw to correct the spelling of 'mulit'
7.	05-11-2001	1. TFLOPS tfallocator and showmesh 2. /home/SANDIA_EVAL/TESTS/ALLOC/sandia_alloc.tar.Z	<untar_dir>/Auto/cmd/tfallocator/yod_base_err	New yod error msg inserted into file 'yod_base_err.ksh'; error msg is generated by command yod -base then cutting and pasting error msg into file, with obvious slight mods made to first line (see file)
8.	05-11-2001	1. Cougar/Yod-Fyod 2. /home/SANDIA_EVAL/TESTS/COUGAR/sandia_cougar.tar.z	<untar_dir>/Auto/cougar/fyod/fy_tester	In file 'fy_tester.ksh', changed rmdir /pfs/fy_tester to rm -rf /pfs/fy_tester so that the command would not hang due to a non-empty directory
9	05-11-2001	1. Cougar/Yod-Fyod 2. /home/SANDIA_EVAL/TESTS/COUGAR/sandia_cougar.tar.z	<untar_dir>/Auto/cougar/hello_hetero	In file 'RUN_p3_htest3', the default base size -base 2 was changed to -base 5 as default for Basil
10.	05-11-2001	1. Cougar/Yod-Fyod 2. /home/SANDIA_EVAL/TESTS/COUGAR/sandia_cougar.tar.z	<untar_dir>/Auto/cougar/hello_hetero	In file 'p3_htest13', the defaults were changed to 2x4x4 yod -p 3 -sz 1x4x4:0,0 hello1 yod -p 3 -sz 1x4x4:0,1 hello2 as default for Basil

11.	05-11-2001	1. Cougar/Yod-Fyod 2. /home/SANDIA_EVAL/TESTS/COUGAR/sandia_cougar.tar.z	<untar_dir>/Auto/cougar/yod/share/sleep_test	In files 'equal.sh' and 'unequal.sh', the default base size -base 2 was changed to -base 5 as default for Basil

12.	05-11-2001	1. Cougar EATs
		
		2. /home/SANDIA_EVAL/TESTS/EATS/sandia_eats.tar.Z
			<untar_dir>/Cougar/perfmon	In file 'eat.auto', the statement
				if [$difference -le 1 -a $difference -ge 0]; then
				was changed to
				if [$difference -le 10 -a $difference -ge 0]; then
				due to timing issues with faster processors

13.	05-11-2001	1. Cougar EATs
		
		2. /home/SANDIA_EVAL/TESTS/EATS/sandia_eats.tar.Z
			<untar_dir>/Cougar/apps/Nodeperf/nodeprf	In file nodeperf.F, change all statements similar to
				if ((...) .gt. 0.5)then
				to
				if((...) .gt. 1.0)then
				since some machines are slower than others (due to single system image) and the original tolerance needed relaxing

14.	05-11-2001	1. Cougar EATs 2. /home/SANDIA_EVAL/TESTS/EATS/sandia_eats.tar.Z	<untar_dir>/Cougar/apps/BLACS/BLACS	In file 'run_blacs_test_long', change set LOCAL = " -comm 2M -stack 30000000 " to set LOCAL = " -comm 25M -stack 30000000 " to increase communications buffer size

15.	05-11-2001	1. VSX Standard
		
		2. /home/SANDIA_EVAL/TESTS/UNIX/sandia_vsx_std.tar.Z
			/home/projects/eval/users/vsx0/SRC/tset/S.POSIX.os/A.files/T.stat	In file 'stat.c', replace the ' - 1' in all statements
				if (stbuf... != t0 - 1)
				and
				if (stbuf... != t1 - 1)
				with ' - 5' to relax tolerances and account for various system sizes

16.	05-11-2001	1. Munops 2. /home/SANDIA_EVAL/TESTS/MUNOPS/munops.tar	/home/projects/eval/munops/compute_sat/comtest_long /home/projects/eval/munops/compute_sat/comtest_grnd_bnc	In file 'cougar.sh' in each directory, change comm-size=2M to comm-size=50M to allow large enough communications buffers to prevent failures and SIGPORTALS errors
17.	05-11-2001	1. Munops 2. /home/SANDIA_EVAL/TESTS/MUNOPS/io_munops_for_janus.tar	/home/projects/eval/io_munops/compute_sat/comtest_long /home/projects/eval/io_munops/compute_sat/comtest_grnd_bnc	In file 'cougar.sh' in each directory, change comm-size=2M to comm-size=50M to allow large enough communications buffers to prevent failures and SIGPORTALS errors
18.	05-11-2001	1. Munops 2. /home/SANDIA_EVAL/TESTS/MUNOPS/io_munops.tar	/home/projects/eval/io_munops/compute_sat/comtest_long /home/projects/eval/io_munops/compute_sat/comtest_grnd_bnc	In file 'cougar.sh' in each directory, change comm-size=2M to comm-size=50M to allow large enough communications buffers to prevent failures and SIGPORTALS errors

19.	05-18-2001	1. Cougar/Yod-Fyod 2. /home/SANDIA_EVAL/TESTS/COUGAR/sandia_cougar.tar.z	<untar_dir>/Auto/cougar/yod/share/sleep_test	Files 'gold1' and 'gold2' were regenerated by the commands ./equal >> gold_1 ./unequal >> gold_2 to reflect being run on Sandia's Basil; these are compared with files 'scratch1' and 'scratch2', respectively
20.	05-22-2001	1. Cougar/Yod-Fyod 2. /home/SANDIA_EVAL/TESTS/COUGAR/sandia_cougar.tar.z	<untar_dir>/Auto/cougar/fyod/fy_tester	In file 'fy_tester.ksh', changed if [\($pass \) -ge \(`expr $ran - $four_percent` \)] to if [$pass -lt `expr $ran - $four_percent `] to properly compare values for a valid test; Also, revision of 'if-then-else' logic was necessary.
21.	06-01-2001	1. Core EATs 2. /home/SANDIA_EVAL/TESTS/EATS/sandia_eats.tar.Z	<untar_dir>/Core	The 'raid' directory permissions were changed to 777; file 'eat.auto' wanted 'raid.sh' to write as 'root' but could not since 'root' equivalences to 'other' on NFS mounted disks
22.	06-01-2001	1. Core EATs 2. /home/SANDIA_EVAL/TESTS/EATS/sandia_eats.tar.Z	<untar_dir>/Core/raid	Files 'eat.auto' and 'raid.sh' were modified to pass the variable $RESULTS correctly
23.	06-01-2001	1. Core EATs 2. /home/SANDIA_EVAL/TESTS/EATS/sandia_eats.tar.Z	<untar_dir>/	The directory 'Results', with subdirectories 'Core' and 'Cougar', were added to the tar file; sub-subdirectories under these were added to mirror the 'Core' and 'Cougar' directories under /home/sigeval/Accept_x86; this allows the 'raid' and other tests to write to proper directories and avoids error messages relating to 'ftp fuji' generated from the file '/home/sigeval/Accept_x86/Libs/EatFuncts.sh'
24.	06-05-2001	1. Parallel APPS 2. /home/SANDIA_EVAL/TESTS/PAPPS/sandia_papps.tar.Z	<untar_dir>/Auto <untar_dir>/Auto/libc	In file runall.papps, for the p3 mode tests, the lines ddt -d testlist.pass gatherres -suffix PcgrRnx${NNODES} -testlist testlist.pass were changed to ddt -d testlist.pass.p3 gatherres -suffix PcgrRnx${NNODES} -testlist testlist.pass.p3 and the file 'testlist.pass.p3' was added to the libc directory. This file was generated simply to avoid running the test message/nx/globtome which is listed in file /home/SANDIA_EVAL/RESULTS_HISTORY/papps/libc/R3.3_36p3x as N message/nx/globtome don't run in p3 mode

25.	06-05-2001	1. MPI 2. /home/SANDIA_EVAL/TESTS/MPI/sandia_mpi.tar.Z	<untar_dir>/MPITEST/Test	In file 'runall.mpi', changed vsu root -c "cp /cougar/lib/puma/libmpi.a /cougar/lib/puma/libmpi.a.orig" to vsu root -c "cp -p /cougar/lib/puma/libmpi.a /cougar/lib/puma/libmpi.a.orig" which includes the -p option to preserve file dates. Also in file 'runall.mpi', changed vsu root -c "/cougar/lib/puma/libmpi.a.orig /cougar/lib/puma/libmpi.a" to vsu root -c "cp -p /cougar/lib/puma/libmpi.a.orig /cougar/lib/puma/libmpi.a" to add the -p option and to correct the obvious error that the cp command was left out of the original statement. This could result in the libmpi.a-type files getting overwritten with the wrong files.

26.	06-11-2001	1. SOCKETS
		
		2. /home/SANDIA_EVAL/TESTS/NETWORK/sandia_sockets.tar.Z
			<untar_dir>/Auto/sockets/inet/ruserok	Two corrections were made.
				1. Although Basil, Ginger, and Rosemary are listed as valid machines for this test, only Basil had the required file
				/etc/hosts.equiv
				So this file was copied over to Rosemary's and Ginger's /etc directory
				2. In the file
				node.c
				the following line
				sprintf(rhosts, "/%s/%s/%s", "home", l_user, hosts);
				was changed to
				sprintf(rhosts, "%s/%s/%s", "/Net/usr/home", l_user, hosts);
				so that the .rhosts file could be found in the user's home directory.

27.	06-11-2001	1. SOCKETS 2. /home/SANDIA_EVAL/TESTS/NETWORK/sandia_sockets.tar.Z	<untar_dir>/Auto/sockets/cmd/netstat	In the file netstat.ksh all references to the interface eep0 were changed to lo0 since the former does not exist on Sandia's eval systems. For more information, type man netstat and read the information regarding the 'Interface' parameter. Also, see the file R3.0 in the directory /home/SANDIA_EVAL/RESULTS_HISTORY/network/sockets/R3.0 for more info from Intel referencing this change.

28.	06-18-01	1. EATS 2. /home/SANDIA_EVAL/TESTS/EATS/Core/unix	<untar_dir>/Core/unix	In the file eat.auto the following two lines (AlarmClock $$ $unix_MAXTIME &) & alarmPid=$! were changed to the following one line alarmPid=`( ( AlarmClock $$ $unix_MAXTIME & echo $! ) & ) | head -1` since t finished. It hung around for one hour before returning a command prompt to user. The problem was that the eat.auto script was not capturing the correct pid for the AlarmClock.
29.		1. AUTODEBUG 2. /home/SANDIA_EVAL/TESTS/TOOLS/sandia_autodebug.tar.Z	<untar_dir>/AutoDebug/cases	File RUNABLE was modified so that the following 5 tests were moved to the end of the file to run last. The tests are known to hang intermittently, leaving jobs running which prevents subsequent tests from loading:
30.	09-20-01	1. EATS 2. /home/SANDIA_EVAL/TESTS/EATS/Core/unix	<untar_dir>/Core/unix	After a system-wide group change to wg-intel, and after permissions were changed on some directories, we found that the sub-test 'at' would not PASS. Later, we discovered that the permissions for the directories /intel-swe and /intel-swe/sigeval on sasn100 had been changed to 770. For some reason, this would not let the 'at' test PASS. A change of permissions to 775 for both directories cured the problem. The 'at' command in /usr/bin has a sticky bit set that forces the command to run as root. Across networks, in this case between the local directories on Basil and the mounted directories on sasn100, the permissions resort to world/other. With no permissions set on world/other for /intel-swe and /intel-swe/sigeval, the test would not run

31.	09-25-01	1. All suites that run ddt, eatnrun, run.fileio, run, runall.mpi, mpitest 2.Most tar files	most tests	This is not a bug fix, but an addition to the output files. The script /home/sigeval/bin/status_checker outputs the status of the system under test. It is now called from any eval suite calling /home/sigeval/bin/ddt or /home/sigeval/bin/eatnrun Other files that were changed to execute the script were .../io/run for the sandia_io_other tests .../MPITEST/bin/mpitest for the MPI tests .../MPITEST/Test/runall.mpi for the MPI tests Note that if sub-tests are run individually using eat.auto, ksh, or some other local script, the script 'status_checker' will not be called. This is due to the sheer number of sub-test scripts that would have to be changed. In this case, it is advised to run the 'status_checker' script separately before running the sub-test script.
32.	03-12-02	1. AutoUnix 2. /home/SANDIA_EVAL/TESTS/UNIX/sandia_autounix.tar.Z	<untar_dir>/AutoUnix/net/ftp	Files: ftp.sh and ftp1.sh (both are scripts) The standard, non-kerberized version of ftp was renamed to /usr/bin/ftp.osf. This was a change with R4.2.0 The version most users need in our environment is the kerberized version in /usr/local/bin/ftp. Because of their path statement, they were frequently invoking /usr/bin/ftp, which hangs through our firewalls. We were getting a number of emails to janus-help about this. This is why the R4.2.0try1 took so long to install. One work-around is to substitute-all of /usr/bin/ftp to /usr/bin/ftp.osf in the scripts 'ftp.sh' and 'ftp1.sh'. A more elegant way is to add FTPCMD=/usr/bin/ftp.osf at the beginning of the scripts and substitute every occurrence of /usr/bin/ftp with $FTPCMD. (The latter way was chosen.)

33.	03-13-02	1. EATS 2. /home/SANDIA_EVAL/TESTS/EATS/sandia_eats.tar.Z	<untar_dir>/Cougar/AutoTprof/perfmon	File: log.gold and reference.log Note: log.gold is static; reference.log is dynamic and changes for each eval; the files must match to PASS an eval The file /usr/include/perfmon.h was modified for R4.2.0. All changes are cosmetic, but the EATS/Tools/perfmon test fails because the reference file for the perfmon test is based on the old /usr/include/perfmon.h. Change Cougar/AutoTprof/perfmon/log.gold to match the current reference.log so that future evals have a match between the two files.
34.	03-13-02	1. AutoTprof 2. /home/SANDIA_EVAL/TESTS/TOOLS/sandia_autotprof.tar.Z	<untar_dir>/AutoTprof/libperfmon/basic/perfmon	Fix is identical to one immediately above (#33) re: files log.gold and reference.log.
35.	05-01-02	1. Parallel Apps 2. /home/SANDIA_EVAL/TESTS/PAPPS/sandia_papps.tar.Z	<untar_dir> Auto	Initially, failed tests included message/nx/gsend_gsum (failed in 3 modes) libc/misc/perror_nodes in P2 mode (expected failure) libc/misc/mynode in P3 mode (expected failure) These tests passed on re-run, but it took a long time to figure out if they failed in p0 or p3 mode, and whether -munix or -masync was set. As a result of the pain of these re-runs, I significantly re-wrote the web page to reflect a different procedure and to better explain the different run modes. Also, the output of the file runall.papps now reflects in which mode the tests are running. Hopefully, some of the mystery behind running this test has been removed, and it is much easier to re-run failures.
36.	05-01-02	1. ddt file i/o 2. /home/SANDIA_EVAL/TESTS/FILEIO/sandia_io.tar.Z	<untar_dir>	The tests Auto/cmd/pfs_munix/lsizecmd_err Auto/cmd/pfs_masync/lsizecmd_err originally failed, probably due to necessity of being logged in when these tests are run; appear to need interactive TTY; will move these to top of test and make some more mods so that these tests can run at the beginning of this eval.

37.	05-01-02	1. AutoUnix 2. /home/SANDIA_EVAL/TESTS/UNIX/sandia_autounix.tar.Z	<untar_dir>/AutoUnix	The following test failed but passed on interactive re-run: arp This is a common failure. I did an extensive test on this suite to separate tests that can be run interactively and in background. Also, the longest test 'netstat' is now run separately since it takes about an hour to complete while other tests run much more quickly. This will prevent 'netstat' from giving the entire test suite a failure when it fails and all else passes. The failed 'arp' test passed on interactive re-run so it was moved to the interactive tests.
38.	05-01-02	1. AutoDebug 2. /home/SANDIA_EVAL/TESTS/TOOLS/sandia_autodebug.tar.Z	<untar_dir>/AutoDebug/cases	Separated all tests alphabetically, then ran each set in background. Sue's filters were used, as well as the latest tar file per Benner. All tests passed except those noted in the web pages. This is the first clean run of AutoDebug I've had in a long time. There's hope for this test yet.
39.	05-01-02	1. AutoTprof 2. /home/SANDIA_EVAL/TESTS/TOOLS/sandia_autotprof.tar.Z	<untar_dir>/AutoTprof	Some tests failed; Sue examined failures which have been addressed by Benner's new 'gold' files that have been checked into CVS. For example, one test spent 10% of time in a routine instead of the expected 7.9%. This discrepancy caused a mismatch in a format statement leading the test to inappropriately conclude a failure.
40.	12-04-02	1. AutoUnix 2. /home/SANDIA_EVAL/TESTS/UNIX/sandia_autounix.tar.Z	<untar_dir>/AutoUnix/net/arp	The 'arp' test has failed intermittently; the problem was found and fixed.

Appendix E: Key Variables Used for the ddt Shell

Variable	Description	Where set	Note
BINDIR	Path of sigeval/bin directory: .../bin	ddt	1
T92LIB	Path of /proj/tfeval/TF[year]/[release]/lib directory: .../t92/lib	ddt	1
FAILFILE	File of expected ddt failures, if any	ddt	1,2,5
MKPART	Indicator whether to make a ddt partition	ddt	?
PARTNAME	Unique mesh partition name: "`hostname`$$`tty`"	ddt	1
PULSEFILE	/tmp/pulse$$	ddt	1,5
	The following are the hardware & software configuration variables		
COMP_OPT	Compiler option: [cx]	ddt	3,5
HOST_ARCH	Host architecture: [PS4]	ddt (hostSys)	3,5
HWTYPE	Hardware type: [GM]	ddt	3,5
N3480	Number of 3480 tapes	ddt (ddtenv.sh)	1,3
NDISK	Number of hard disks	ddt (ddtenv.sh)	1,3
NEXAB	Number of ExaByte tapes	ddt (ddtenv.sh)	1,3
NIO	Number of I/O nodes	ddt (ddtenv.sh)	1,3
SNODE	Number of service nodes	ddt (ddtenv.sh)	1,3
STIME	Sleep time for SRD	ddt (ddtenv.sh)	1,3

ddt Files	Description	Default value 4
FAILFILE	File of expected ddt failures, if any	
PULSEFILE	/dev/null (default) or /tmp/pulse$$ (-s option)	
RESTARTFILE	File of all previous ddt runs, by host, tty & user This file constantly grows, and is not removed.	

Makefile Macros	Description	Default value 4
AT	Macro for "@" symbol, used for echoing	@
AR	Paragon "native" archiver command	ar
AR860	Cross-development archiver command	ar860
CC	Paragon "native" C compiler command	cc
ICC	Cross-development C compiler command	icc
F77	Paragon "native" Fortran compiler command	f77
IF77	Cross-development Fortran compiler command	if77
BUF	Optional message passing buffer	
CFLAGS	Optional C compiler flags	
CONCUR	Compile sources for multi-processing	-Mconcur
DEFHOST	Host compiler command line defines	-DHOSTPROG
FFLAGS	Optional Fortran compiler flags	
HOSTLIBS	Host compiler library	
MLIB	Math library macro	-lm
NXFLAG	Compile sources for auto mesh starting	-nx
PXFLAG	Compile sources for manual mesh starting	-lnx
CLIB	Eval common C library	$(T92LIB:lib=)Auto/libc/lib
FLIB	Eval common Fortran library	$(T92LIB:lib=)Auto/libf/lib
LOCALLIB	Local library macro	
COMPILE_RULE	General makefile rule for compile targets	(not listed)
EXEC_RULE	General makefile rule for executable targets	(not listed)
LOG_RULE	General makefile rule for log targets	(not listed)
OBJ_RULE	General makefile rule for object targets	(not listed)

Makefile Files	Description	Default value 4
FAILFILE	File of expected ddt failures, if any	
PULSEFILE	/dev/null (default) or /tmp/pulse$$ (-s option)	

```
Optional
Variable    Description 4
---------   ----------------------------------------------
     The following are the partition management and application variables
APPL_ARGS   ??
EPL         Partition's effective priority limit
MKPART      Indicator whether to make a ddt partition or not
MOD         Partition's permissions
MSG_OPTS    ??
ND          Nodes to allocate in partition
NNODES      Number of nodes in application/mesh             ddt (ddtenv.sh)  1,3
NT_APPL     Node type application flag
NT_PART     Node type application flag
ON          Node list for application
PARTNAME    Unique mesh partition name
PRI         Application's priority value
PT          Application's ptype
RLX         ??
RQ          Rollin quantum
SS          Standard scheduled partition
SZ          Number of nodes to allocate in partition/application

     The following are the message passing variables replaced by MSG_OPTS
GTH         Give threshold           +
MBF         Memory buffer            |
MEA         Memory each              | Per application optional
MEX         Memory export            | message passing customization
PKT         Packet size              | switches
PLK         Process lock             |
SCT         Send count               |
STH         Send threshold           +
```

Notes:

1. Set by ddt, if not already defined.

2. ddt command line option.

3. Host and mesh related variables, available for run scripts.

4. ddt arguments that are passed to the make command.

5. Used by both ddt and the master Makefile.

Appendix F: EATS web pages

Topics:

Target Machines

As described in the
Test Matrix, 90% Confidence Level, 2-Week Eval
or the
Test Matrix, 3-Week Complete Eval,
the target machines are:

- basil

The **Xdebug Manual** test described after the **EATs** tests can be run on any eval machine.

Environmental Setup

NOTE: The TFLOPS EATS only run under **csh**. If your login shell is **sh** or **ksh**, you must change shells to **csh** before executing the EATS on the TFLOPS system.

As **root**, on your **TFLOPS-system-under-test**:

1. Contact your system administrator to install desired **TOS** software, preferably in such a way as to "clean" the entire system of old/out-of-date software (also called a "full" install).

2. Verify that the following users are installed on the **TFLOPS-system-under-test**:
   - The test runner (yourself)
   - tcptest (used by the Core/tcp-ip EAT)
   - nfstest (used by the Core/nfs EAT)

   The best way to check is as follows:
   ```
   grep tcptest /etc/passwd
   grep nfstest /etc/passwd
   ```
3. Verify that the evaluation directory structure is present. If not, contact your system administrator to mount the necessary NFS file systems.
   ```
   ls -ld /home/sigeval
   ```

4. OPTIONAL NQS SETUP: do this only if you want to run the **NQS**. **EAT NQS** setup information

is detailed on the **NQS Setup for EATS** web page.

As **EATS-runner (yourself)**, on your **TFLOPS-system-under-test**:

Untar sandia_eats.tar.Z in /home/sigeval/Accept_x86_R?_? as follows:

```
mkdir /home/sigeval/Accept_x86_R?_?
cd /home/sigeval/Accept_x86_R?_?
zcat /home/SANDIA_EVAL/TESTS/EATS/sandia_eats.tar.Z | tar xpf - &
        (require 20 minutes to zcat and untar)
```

Verify that **/home/sigeval/bin** exists. If not, cd to **/home/sigeval** and untar **/home/SANDIA_EVAL/TESTS/sigeval_bin.tar**, which will create the bin directory and populate it, as follows:

```
cd /home/sigeval
tar xvf /home/SANDIA_EVAL/TESTS/sigeval_bin.tar
```

NOTE: you may get lots of statements like

```
tar: can't create bin/.graveyard/delta_env.5.gz: Permission denied
```

Ignore these, as the proper files will be untarred to **/home/sigeval/bin**.

Verify that **/usr/local/bin/vsu** exists and has **4750 permissions** with owner of **root** and group of **chgrp wg-intel**. This file is used by the 'nfs', 'tcp-ip', 'raid', 'xtrnl', and 'allocator' tests, possibly others.

If not, compile **vsu.c** source and (conditionally) set permissions & ownership. The user must first copy the file **vsu.c** into a temporary directory as **test-runner (yourself)**, then **su** as **root** and **copy vsu.c** into **/usr/local/bin** before compiling, as follows:

As **test-runner (yourself)**, on your **TFLOPS-system-under-test**:

```
cp /home/sigeval/bin/vsu.c <your_temp_directory>
```

As **root**, on your **TFLOPS-system-under-test:**

```
cp <your_temp_directory>/vsu.c /usr/local/bin
cd /usr/local/bin
cc -o vsu vsu.c && chmod u+s vsu && chown root vsu
chgrp wg-intel vsu
exit
```

As **EATS-runner (yourself)**, on your **TFLOPS-system-under-test**:

Add these directories to EATS-runner's path:

```
/usr/local/bin
/cougar/bin
/home/sigeval/bin
/sbin
```

Edit the following file, changing **WW** string to the work week you are testing:

```
vi /home/sigeval/bin/eats.cshrc
```

NOTE: If this file is not found in this location, then **sigeval_bin.tar** needs to be untarred in the appropriate directory, as described above.

Set the environmental variables by executing the following. The variables set here are used for running the EATS shown later on this page. Please view "**eats.cshrc**" for a complete list of the variables and what they represent:

```
source /home/sigeval/bin/eats.cshrc
```

Create a results file with the following one-line command (**$Ax** is defined in the **eats.cshrc** file, above):

```
cp /home/sigeval/bin/EAT_results_template $Ax/$WWthis-x86-EAT-results
```

Chart for Running the EATS

DO NOT CUT AND PASTE COMMANDS FROM THIS CHART.
THIS CHART IS FOR ILLUSTRATION ONLY!

EAT type		Cut and Paste pieces for running EATs and add saving to file
CORE (OSF) EATs:	Sigeval's alias to run all eats	$ENRx -P $Ax $Cx/allocator $Cx/manpage $Cx/nfs $Cx/pmake $Cx/raid $Cx/sat $Cx/tcp-ip $Cx/unix $Cx/xtrnl $Cx/nqs $Cx/macs <<< save to tee -a $Ax/$WWthis.x86.results.log. >>>
	Eat Invocation Pieces	$ENRx -P $Ax $Cx/allocator $Cx/manpage $Cx/nfs $Cx/nqs $Cx/macs $Cx/pmake $Cx/raid $Cx/sat $Cx/tcp-ip $Cx/unix $Cx/xtrnl save to file with tee -a $Ax/$WWthis.x86.results.log.
Cougar EATs:	Sigeval's alias to run all eats	$ENRx -P $Ax $Cgrx/apps $Cgrx/basic $Cgrx/fileio $Cgrx/message $Cgrx/parallel $Cgrx/pfs $Cgrx/mpi $Cgrx/benchmark $Cgrx/scalapack $Cgrx/perfmon $Cgrx/blas $Cgrx/fft <<< save to tee -a $Ax/$WWthis.x86.results.log. >>>
	Eat Invocation Pieces	$ENRx -P $Ax $Cgrx/apps $Cgrx/basic $Cgrx/fileio $Cgrx/benchmark $Cgrx/message $Cgrx/parallel $Cgrx/pfs $Cgrx/mpi $Cgrx/scalapack $Cgrx/perfmon $Cgrx/blas $Cgrx/fft save to file with tee -a $Ax/$WWthis.x86.results.log.
Tools EATs:	Sigeval's alias to run all eats	$ENRx -P $Ax $Cgrx/AutoPMT $Cx/dbmalloc $Cgrx/cop $Cx/debug <<< save to tee -a $Ax/$WWthis.x86.results.log. >>>
	Eat Invocation Pieces	$ENRx -P $Ax $Cgrx/AutoPMT $Cx/dbmalloc $Cgrx/cop $Cx/debug save to file with tee -a $Ax/$WWthis.x86.results.log.

Login to the **TFLOPS-system-under-test** as **yourself**. Remember that your environment must be set up to run the **EATS**.

From a **csh** shell execute

```
vi /home/sigeval/bin/eats.cshrc (change WW if not done already)
source /home/sigeval/bin/eats.cshrc
```

Some of the environmental variables set up by the files in **EATs-config**, are **$ENRx** which is the run script, and **$Cx** and **$Cgrx** which are directories under the top directory of **Accept_x86_R?_?**. The **EAT** tests are under **$Cx** and **$Cgrx**. Each test can be run by itself by omitting all the other tests from the command line. In addition, the **$ENRx** run-script driver can be bypassed by **cd**'ing into each test directory (e.g., unix, nfs, etc.) and running the tests via the run-script **eat.auto** with a modified version of the file **testlist**. This is often necessary when debugging failing tests.

To create necessary links in the **/apps**, **/message**, **/parallel**, **/benchmark**, **/cop**, and **/signal** test directories, execute the following:

```
cd /home/sigeval/Accept_x86_R?_?/Cougar/util
install_link (needed once per Accept_x86_R?_? test directory)
cd ../..
```

As **root**:

As **root**, add yourself as queue manager if you are not already.

```
qmgr
        Mgr: show manager
        Mgr: add manager your_username:m
        Mgr: exit
exit (exit from being root)
```

Use the **add manager** command only if you are not listed under **show manager**.

Example for adding username **joe**:

```
add manager joe:m
```

where **joe** is to be replaced by your **username**.

Go to the **EATS command lines** below.

NOTE: If for any reason you need to stop one of the following **$ENRx** invocations, do -NOT- use **control-C**, instead, open another window to the **TFLOPS system-under-test** and **kill** the **eatnrun** process. This should kill all the tests in the test list.

EATS command lines CORE, COUGAR, and TOOLS

The EATS have 4 sections:

1. Core (1 run, 10 tests; tests the EAGLE service nodes)
2. Cougar (2 runs, 12 tests per run, 14 hours per run; tests the Kestrel nodes)
3. Tools (1 run, 4 tests; tests parts of Cougar OS)
4. Manual (2 runs -- considered separately from the first 3; tests Cougar nodes; must be run interactively)

Below you will find 3 commands, which run the Core, Cougar, and Tools EATS.

With the advent of **Virtual Nodes**, the **Cougar EATS** command is also run in **-p 3** mode. This invokes the additional processor on each node. See **yod man page** for a complete discussion of **-proc 3** mode.

Finally, you will find the two **Manual** test commands below. These are considered separately since an interactive window will need to be displayed back to the client.

To summarize, there are a total of 6 commands that need to be run to complete the EATS:

- 3 command lines, one each for the Core, Cougar and Tools EAT
- 1 command line for the "-p 3" virtual node Cougar EAT
- 2 manual command lines for the Manual EATS.

NOTE: The following commands are one command line and the log file name ends with a "."

NOTE: the **raid EAT** depends on **/dev/io1/scsi0** existence, and will fail if that device is not present.

If tests fail or hang, either re-run the entire test or re-run the tests individually by running **eat.auto** in the test directory (e.g., unix, nfs, etc.) and modifying the file **testlist**.

I. To run **Core EATS**:

WARNING: If the following permissions do not conform EXACTLY as stated, the " Permissions/Ownership Check" check will fail in the NQS sub-test causing it to fail.

Ensure the file '**/usr/spool/nqs/conf/sched_param**' has '**root**' as owner and '**wg-intel**' as group:

```
ls -l /usr/spool/nqs/conf/sched_param
        su (login as root, if necessary)
        chown root /usr/spool/nqs/conf/sched_param (if necessary)
        chgrp wg-intel /usr/spool/nqs/conf/sched_param (if necessary)
        exit
```

and that the following directories have '**root**' as owner and '**wg-intel**' as group:

```
ls -ld /usr/spool/nqs/log.d
        su (login as root, if necessary)
        chown root /usr/spool/nqs/log.d (if necessary)
        chgrp wg-intel /usr/spool/nqs/log.d (if necessary)

ls -ld /usr/spool/nqs/conf
        chgrp wg-intel /usr/spool/nqs/conf (if necessary)
        exit
```

As **root**, make sure that no other **node_groups** or **partitions** exist before you start running the tests. For example, if the EATs were run on the **system_under_test** or an NQS test stopped prematurely, you may find a **node_group** such as **eats_node_group** and/or **qmgr_node_group**, or a partition named **OPEN**. Remove them as follows:

```
su (to login as root)
qmgr
        Mgr: show node
        Mgr: delete node_group eats_node_group
        Mgr: delete node_group qmgr_node_group
        exit
lspart
rmpart partition_name (if necessary)
exit
```

As root, make sure the following link exists in the /home directory:

```
sigeval -> /Net/intel/intel-swe/sigeval
```

If not, then issue the following command in the /home directory

```
ln -s /Net/intel/intel-swe/sigeval sigeval
```

As yourself, source the following file, and run **install_link**, if not done already:

```
source /home/sigeval/bin/eats.cshrc
unsetenv COP_OPT

cd /home/sigeval/Accept_x86_R?_?/Cougar/util
install_link (needed once per Accept_x86_R?_? test directory)
cd ../..
```

As **yourself**, execute the following one-line test and remain logged in while it is executing. You must be logged in for these tests (**unix**, **nfs**, and **tcp-ip**) to run. The '**unix**' test performs some interactive tests on the current window, for example. Do **not** log out while these 3 test suites run. Do **not** redirect output.

```
$ENRx -P $Ax $Cx/unix $Cx/nfs $Cx/tcp-ip $Cx/nqs
| & tee -a $Ax/$WWthis.x86.results.log.core_eats_1.machine_user_date
        (approx. times:
        7 minutes to run unix;
        3 minutes for nfs;
        5 minutes for tcp-ip;
        Total Time: 16-18 minutes)
```

Once the above test finishes, execute this one-line command to run 7 test suites:

```
$ENRx -P $Ax $Cx/allocator $Cx/manpage
$Cx/pmake $Cx/raid $Cx/sat $Cx/xtrnl
  > $Ax/$WWthis.x86.results.log.core_eats_2.machine_user_date &
                (requires 1 hour, 40 minutes on Basil)
```

Optionally, the last line above can be replaced with

```
| & tee -a $Ax/$WWthis.x86.results.log.core_eats_2.machine_user_date
```

Always check for tests that did not run, as well as test failures, by executing the following commands:

```
grep 'TEST_RESULT'
$Ax/$WWthis.x86.results.log.core_eats.machine_user_date
grep 'NOT_RUN'
$Ax/$WWthis.x86.results.log.core_eats_2.machine_user_date
grep 'FAILED' $Ax/$WWthis.x86.results.log.core_eats_2.machine_user_date
```

II. To run **Cougar EAT**s (CAN BE RUN SIMULTANEOUSLY WITH **CORE EAT**s!):

Source the following file, and run **install_link**, if not done already:

```
source /home/sigeval/bin/eats.cshrc
unsetenv COP_OPT

cd /home/sigeval/Accept_x86_R?_?/Cougar/util
install_link (needed once per Accept_x86_R?_? test directory)
cd ../..
```

and execute this one-line command to run 12 test suites:

```
$ENRx -P $Ax $Cgrx/pfs $Cgrx/apps $Cgrx/basic $Cgrx/message
$Cgrx/parallel $Cgrx/mpi $Cgrx/benchmark $Cgrx/scalapack
$Cgrx/perfmon $Cgrx/blas $Cgrx/fft $Cgrx/fileio
  > $Ax/$WWthis.x86.results.log.cougar_eats.machine_user_date &
                (requires 2 hrs, 55 min on Basil)
```

Optionally, to append results and have output directed to the screen simultaneously, the last line above can be replaced with

```
| & tee -a $Ax/$WWthis.x86.results.log.cougar_eats.machine_user_date
```

Always check for tests that did not run, as well as test failures, by executing the following commands:

```
        grep 'TEST_RESULT'
$Ax/$WWthis.x86.results.log.cougar_eats.machine_user_date
        grep 'NOT_RUN'
$Ax/$WWthis.x86.results.log.cougar_eats.machine_user_date
        grep 'FAILED' $Ax/$WWthis.x86.results.log.cougar_eats.machine_user_date
```

III. To run **Tools EATs**:

Source the following file, and run **install_link**, if not done already:

```
        source /home/sigeval/bin/eats.cshrc
        unsetenv COP_OPT

        cd /home/sigeval/Accept_x86_R?_?/Cougar/util
        install_link (needed once per Accept_x86_R?_? test directory)
        cd ../..
```

and execute this one-line command to run 4 test suites:

```
        $ENRx -P $Ax $Cgrx/AutoTprof $Cx/dbmalloc $Cgrx/cop $Cx/debug
          > $Ax/$WWthis.x86.results.log.tools_eats.machine_user_date &
                        (requires 30 minutes on Basil)
```

Optionally, to append results and have output directed to the screen simultaneously, the last line above can be replaced with

```
        | & tee -a $Ax/$WWthis.x86.results.log.tools_eats.machine_user_date
```

Always check for tests that did not run, as well as test failures, by executing the following commands:

```
        grep 'TEST_RESULT'
$Ax/$WWthis.x86.results.log.tools_eats.machine_user_date
        grep 'NOT_RUN' $Ax/$WWthis.x86.results.log.tools_eats.machine_user_date
        grep 'FAILED' $Ax/$WWthis.x86.results.log.tools_eats.machine_user_date
```

IV. To run **Cougar EATs** in **-p 3** mode:

This tests Virtual Node code (see **yod man page** and discussion of **-proc 3** mode).

Source the following file, and run **install_link**, if not done already:

```
        source /home/sigeval/bin/eats.cshrc

        cd /home/sigeval/Accept_x86_R?_?/Cougar/util
        install_link (needed once per Accept_x86_R?_? test directory)
        cd ../..
```

Set the environment variable **COP_OPT** to "**-p 3**":

```
        setenv COP_OPT "-p 3"
```

After setting **COP_OPT** to "**-p 3**", execute the **$ENRx** one-line command to run 12 test suites:

```
        $ENRx -P $Ax $Cx/allocator $Cgrx/apps $Cgrx/basic $Cgrx/fileio
        $Cgrx/message $Cgrx/parallel $Cgrx/pfs $Cgrx/mpi $Cgrx/benchmark
        $Cgrx/scalapack $Cgrx/perfmon $Cgrx/blas
          >  $Ax/$WWthis.x86.results.log.cougar_eats_p3.machine_user_date &
                        (requires 3 hrs, 20 min on Basil)
```

Optionally, to append results and have output directed to the screen simultaneously, the last line above can be replaced with

```
        | & tee -a $Ax/$WWthis.x86.results.log.cougar_eats_p3.machine_user_date
```

Always check for tests that did not run, as well as test failures, by executing the following commands:

```
        grep 'NOT_RUN'
$Ax/$WWthis.x86.results.log.cougar_eats_p3.machine_user_date
        grep 'FAILED'
$Ax/$WWthis.x86.results.log.cougar_eats_p3.machine_user_date
```

This test has been successfully run on the following **Basil** and **Rosemary**. It is expected to run on any of the eval machines.

If the test runner used ssh to login to TFLOPS system-under-test, then skip to sourcing the **eats.cshrc** file. **Note that if ssh is used at Sandia, the user should make sure the 'Forward X11' is checked in the 'Connection Tab' that appears when clicking on 'Properties' in the ssh window used to connect to the remote machine.** There is no need to **xhost +** and **setenv DISPLAY** as shown below, or to ensure the **IP address** and **system name** is in the **/etc/hosts** file on the **TFLOPS** system.

However, if the test runner did not **ssh** in to the **system-under-test**, follow the steps detailed below.

The system that will display the **Xdebug** windows must have it's **IP address** and **system name** in the **/etc/hosts** file on the **TFLOPS** system.

On the system that will display the **Xdebug** windows, execute

```
/usr/X11R6/bin/xhost +
setenv DISPLAY "your host machine":0
```

to allow the **Xdebug** windows to be opened.

SKIP TO HERE IF USING **SSH** AT SANDIA:

Source the **eats.cshrc** file. From a **csh** shell execute

```
source /home/sigeval/bin/eats.cshrc
unsetenv COP_OPT
```

Ensure no **LOCKED** file exists in the following directory.

```
rm /home/sigeval/Accept_x86_R?_?/Core/xdebug/LOCKED
```

Start the **X Server** on the local machine, if necessary.

To begin the **Xdebug EAT**, execute

```
cd /home/sigeval/Accept_x86_R?_?
```

and the one-line command (do **NOT** run in background, and do **NOT** redirect output!)

```
$ENRx -P $Ax $Cx/xdebug
 | & tee -a $Ax/$WWthis.x86.results.log.xdebug_eats.machine_user_date
```

The test begins by bringing up the **gray debugger window**, a process that may take **3-5 minutes** when using dialup. The original window in which you are working will display the question

INSTRUCTION: Wait for xdebug to come up.
EXAMINE: Make sure the xdebug came up properly.
Did the proper results occur? [Y/N]

before the debugger window appears, so be patient. Any other message than above probably means you did not start the X Server on your local machine. It may also mean that settings are not correct on your X Server for communicating with the remote machine.

This test requires approximately 10-15 minutes for the user to complete.

The display you will see in a window separate from the debugger window is shown below. The numbers are used for reference on the web page and are not displayed on the screen.

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

=> Xdebug EAT started: Wed Nov 18 13:14:41 PST 1998 looking for
include/X11 directory found include/X11 directory
Please enter the name of the host to display the test on:
1. INSTRUCTION: Wait for xdebug to come up.
        EXAMINE: Make sure the xdebug came up properly.
        Did the proper results occur? [Y/N]
2. INSTRUCTION: Select the "Command Line" panel. Enter in the
        lower frame,
                debug -n hello
        and hit return. Return back to the "Command Line" panel.
        Enter in the
        lower frame,
                debug
        and hit return.
        EXAMINE: No Errors should occur and status messages will
        be printed the screen.
        Did the proper results occur? [Y/N]
3. INSTRUCTION: Select the "Input Output" panel. Select "Execute"
        from the top area of the window. Select "Restart".
        EXAMINE: There should be output to the I/O panel.
        Did the proper results occur? [Y/N]
4. INSTRUCTION: Select the "Load" panel. Select "File/Dir List" to
        bring up the "Application Title" window.
        EXAMINE: The "Application Title" window should come up without
problems.
        Select "mpi_ctest" in the Files field. Select "OK" button.
        EXAMINE: The "Application Title" should disappear, and "mpi_ctest"
        should appear in the "To Debug" field.
        Did the proper results occur? [Y/N]
5. Set the number of nodes to 6 or more in the "Mesh Size" panel.
        Select the "Debug" button.
        EXAMINE: Upon the successful load, you should be brought to the
        "Process" panel.
        Did the proper results occur? [Y/N]
6. INSTRUCTION: Go to the "Source" panel. Select "main" in the list at the
        left. Click in the "action point column" , the column with the
        triangular shapes next to the source code, to set an action point
        at line 53.
        EXAMINE: You should have been able to set the action point.
        Did the proper results occur? [Y/N]
7. INSTRUCTION: Select the "Process" panel, and select "Continue".
        EXAMINE: The Program should stop at the action point on line 53.
        This will show at the bottom of the window.
        Did the proper results occur? [Y/N]
8. INSTRUCTION: Select "Source" panel. Go to line 38. Click on "i"
        with the middle mouse button. If you do not have a middle mouse
        button, click on the two buttons simultaneously.
        EXAMINE: You should now be in the "Data" panel. "i=" should show
        on the screen.
        Did the proper results occur? [Y/N]
9. INSTRUCTION: Select the "Process" panel. Select "Step".
        Select "Step" again.
        EXAMINE: Should be at a line greater than 53.
        Did the proper results occur? [Y/N]
10. INSTRUCTION: Select the "Messages" panel. Select the "Update"
```

```
        button.
        EXAMINE: The nodes should show as gray squares.
        Did the proper results occur? [Y/N]
11. INSTRUCTION: Select the "Process" panel and select "Continue".
        Select "File" from the top area of the window. Select "Exit".
        EXAMINE: Does xdebug exit cleanly.
        Did the proper results occur? [Y/N]

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Signal Semi-Automated Version

Source the following file, and run **install_link**, if not done already:

```
source /home/sigeval/bin/eats.cshrc
unsetenv COP_OPT

cd /home/sigeval/Accept_x86_R?_?/Cougar/util
install_link (needed once per Accept_x86_R?_? test directory)
cd ../..
```

To execute the semi automated version execute

```
cd /home/sigeval/Accept_x86_R?_?/Cougar/signal
eat.int
```

and enter **control C** when prompted. You will be prompted a total of 4 times.

Wrap Up

Edit the file **$WWthis-x86-EAT-results** by adding the work week and summary of passed tests. To add the test summary, search **$Ax/$WWthis.x86.results.log.** for the string **=>** and append the search results to **$WWthis-x86-EAT-results**.

Mail the results file shown above to interested parties. Run the **Manual EATS** before sending out results. Note that **Xdebug** and **Signal** tests do not automatically go into log files. The results of these tests must be put into the final Test Template manually.

To remove files arising from compilations:

```
cd /home/sigeval/Accept_x86_R?_?
find * -name "*.cgr-x" -exec rm {} \; -print
find * -name "*.o" -exec rm {} \; -print
find * -name "*.cgr-c" -exec rm {} \; -print
```

Post-Processing with Mathematica

[This section under development (dwbarnette)]

The results of this eval test can be post-processed using Wolfram Research's *Mathematica*. *Mathematica* programs, also known as 'notebooks', have been developed and written at Sandia to plot the run time and pass/fail data in the form of colored bar charts. The bar charts may be copied to text editors such as Microsoft's Word or to viewgraph editors such as Microsoft's PowerPoint for presentation purposes. It is assumed that the analyst has access to, and is at least somewhat familiar with using, a recent version of *Mathematica*.

First, copy the following files to the target machine's directory in which *Mathematica* will be run:

$T92DIR...
$T92DIR...

Next, copy the following *Mathematica* notebook for this analysis to the same location:

/home/sigeval/Mathematica_Notebooks/...

Run the program to generate the plots.

Known Error Messages -- Ignore These

On most EATS - "./eat.auto: 196781 Terminated" - (Number will vary.)

On several EATS - unable to copy - obsolete commands and system names. The error message below may appear after an EAT test is run. It simply means that the results file was not written to the general results directory, but was instead written to the EAT subtest directory. The error message can be disregarded.

```
Copying results to /home/sigeval/Accept_x86/Results/Core/sat.
Unable to automatically copy the results to the EATs Results
directory
(/home/sigeval/Accept_x86/Results/Core/sat).
Please do it manually via ftp:
    ftp fuji
    put results.out
/home/sigeval/Accept_x86/Results/Core/sat/results.out
    quit
```

On cougar apps EAT - skipped tests - never removed.

```
FORTRAN STOP
S: 1440 TESTS; 1080 PASSED,  360 SKIPPED,    0 FAILED.
```

On scalapack EAT - "cp ... *.dat adm ../*.dat are identical" - (many times)

```
cp: /home/sigeval/Accept_x86/Cougar/scalapack/TESTING/HRD.dat and
                        ../HRD.dat are identical (not copied).
*** Exit 1
Stop.
```

On scalapack EAT - "tests skipped because of illegal input values." (many times)

X windows EAT - (obsolete over-head)

```
=> x EAT started: Mon Dec 14 14:40:11 PST 1998
looking for include/X11 directory
found include/X11 directory

Please enter the name of the host to display the test on: ubirr

Warning: unable to open central results file:
 /home/sigeval/Accept_x86/Results/Core/x
Writing results file to current working directory.
```

Cougar/apps/memtest and **Cougar/apps/nodeperf** may fail on systems that contain Pentium Pro 200Mhz processors, rather than Pentium 333Mhz w/Xeon core tech processors.

Appendix G: Miscellaneous Utility Scripts

Script 1: checkout tests from CVS

```
#!/bin/csh
#
# UNLESS OTHERWISE NOTED:
# This script must be run on tweety.sandia.gov
#    Otherwise, the script will fail
#
# Filename: utility_cvs_checkout_2wk_evals
#
# To run:
# Execution command:
#              utility_cvs_2wk_evals OS_VERSION > logfile_CVS_2wk_evals &

# Monitor logfile for progress and completion

#
# Author: D. W. Barnette
# Date: January, 2003
#

echo ""
echo "    CVS CHECKOUT SCRIPT FOR 2-WEEK EVAL TEST SUITE"
echo "    ============================================="
echo ""
echo " Start time: `date`"
echo ""
# check for correct host
setenv HOSTNAME `hostname`
echo ""
echo "  Host name: $HOSTNAME"

if ($HOSTNAME != "tweety") then
    echo ""
    echo " This script must be run on TWEETY or it will fail."
    echo " Pls login to TWEETY and re-run this script."
    echo " Now exiting."
    echo ""
    date
    echo ""
    exit()
 else
    echo ""
endif

if ( $#argv == 1 ) goto goodargs
echo " "
echo " USAGE ERROR: You must type the name of the "
echo "       OS version (R4_2_2, for example) and re-direct "
echo "       output to a log file! "
echo " "
```

```
echo "   Usage: utility_cvs_2wk_evals   OS_VERSION   >   logfile_CVS_2wk_evals &"
echo " "
echo " "
exit 1

goodargs:

echo " "
echo "   *****  CHECK-OUT THE 2-WEEK EVAL FROM CVS ***** "
echo ""
echo " Starting date/time: `date`"

setenv OS_VERSION $1
echo " "
echo " OS_VERSION = $1 "
echo " "

setenv SIGBINDIR /intel/intel-swe/sigeval/bin
echo " SIGBINDIR = $SIGBINDIR"
# use following for EATS
setenv EVALDIR1 /intel/intel-swe/sigeval/Accept_x86_"$OS_VERSION"
# use following for most tests
setenv EVALDIR2 /intel/intel-swe/sigeval/"$OS_VERSION"
# use following for AutoDebug
setenv EVALDIR3 /intel/intel-swe/sigeval/"$OS_VERSION"_AutoDebug
# use following for Munops
#setenv EVALDIR4 $EVALDIR2/Munops/"$OS_VERSION"
setenv EVALDIR4 /intel/intel-swe/Eval_drop_box
# use following for IO-Munops
#setenv EVALDIR5 $EVALDIR2/IO_Munops/"$OS_VERSION"
setenv EVALDIR5 /intel/intel-swe/Eval_drop_box

echo " EVALDIR1 = $EVALDIR1 (for EATS tests)"
echo " EVALDIR2 = $EVALDIR2 (for most tests)"
echo " EVALDIR3 = $EVALDIR3 (for AutoDebug)"
echo " EVALDIR4 = $EVALDIR4 (for Munops files extraction directory)"
echo " EVALDIR5 = $EVALDIR5 (for IO-Munops files extraction directory)"

echo ""

setenv CVSROOT /data1/os/CVS_Tflops_Eval_Repository
echo " CVS Repository: $CVSROOT"
echo ""

# check on existence of CVS repository
if (! -x $CVSROOT) then
    echo ""
    echo " >> CANNOT FIND CVS REPOSITORY "
    echo " Looking for: $CVSROOT"
    echo "  The repository is needed to run this script,"
    echo "      but it cannot be found."
    echo " Exiting"
    echo ""
    date
    echo ""
    exit()
```

```
endif

# create the sigeval/bin directory if it does not exist
if (! -x $SIGBINDIR) then
    echo ""
    echo " >> Creating directory $SIGBINDIR"
    cd /intel-swe
    mkdir -p $SIGBINDIR
    cvs -d $CVSROOT checkout sigeval &
else
    echo ""
    echo " >> Directory $SIGBINDIR exists"
endif

# if dir still does not exist (for some reason, the dir could not be made), quit
if (! -x $SIGBINDIR) then
     echo ""
     echo " Cannot create $SIGBINDIR"
     echo "  Script exiting!"
     echo ""
     echo ""
     exit
endif

# create the evaluation directories if they don't already exist
if (! -x $EVALDIR1) then
echo ""
echo " >> Creating directory $EVALDIR1"
       mkdir -p $EVALDIR1
else
echo ""
echo " >> Directory $EVALDIR1 exists"
endif

# if dir still does not exist (for some reason, the dir could not be made), quit
if (! -x $EVALDIR1) then
     echo ""
     echo " Cannot create $EVALDIR1"
     echo "  Script exiting!"
     echo ""
     echo ""
     exit
endif


if (! -x $EVALDIR2) then
echo ""
echo " >> Creating directory $EVALDIR2"
       mkdir -p $EVALDIR2
else
echo ""
echo " >> Directory $EVALDIR2 exists"
endif

# if dir still does not exist (for some reason, the dir could not be made), quit
if (! -x $EVALDIR2) then
```

```
    echo ""
    echo " Cannot create $EVALDIR2"
    echo "  Script exiting!"
    echo ""
    echo ""
    exit
endif
 .
 .
<repeat above until all EVALDIRX directories are created>
 .
 .
echo ""
echo ""
echo " > Checkout eval suites from CVS repository"
echo ""
echo " date/time: `date`"



# start loading up directories for eval

# 1. EATS
echo ""
echo " 1. EATS"
echo " Current time/date: `date` "
echo " Requires 20 minutes to checkout of CVS"
echo " Target directory: $EVALDIR1"
setenv EATS_FLAG 0
     cd /intel-swe
     setenv EATS_FLAG 1
     cvs -d $CVSROOT checkout -d $EVALDIR1 EATS &
if ($EATS_FLAG == 0) then
      echo " -- EATS not untarred -- "
endif

# 2. Mini-EATS
echo ""
echo " 2. Mini-EATS"
echo " Current time/date: `date` "
echo " Requires 45 minutes to checkout of CVS"
echo " Target directory: $EVALDIR1"
setenv MINI_EATS_FLAG 0
#      cd $EVALDIR1
#      setenv MINI_EATS_FLAG 1
#      zcat /intel/intel-swe/SANDIA_EVAL/TESTS/EATS/mini_eats.tar.Z | tar xpf -
if ($MINI_EATS_FLAG == 0) then
      echo " -- Mini-EATS not untarred -- "
endif


# 3. Parallel Apps
echo ""
echo " 3. Parallel Apps"
echo " Current time/date: `date` "
echo " Requires 10 minutes to checkout of CVS"
```

```
echo " Target directory: $EVALDIR2"
setenv PAPPS_FLAG 0
      cd $EVALDIR2
      setenv PAPPS_FLAG 1
      cvs -d $CVSROOT checkout                                    \
     Auto/libc/message Auto/libc/misc Auto/libc/global                \
      Auto/libc/touch Auto/libf/message Auto/libf/misc Auto/libf/global   \
      Auto/libc/testlist.pass Auto/libf/testlist.pass Auto/runall.papps   \
      LibBldLst lib Auto/libc/lib Auto/libf/lib                   \
      Auto/libc/testlist.pass.p2 Auto/libc/testlist.pass.p3
if ($PAPPS_FLAG == 0) then
      echo " -- Parallel Apps not untarred -- "
endif
 .
 .
# 4. MPI
.<similar to above; details left out for brevity>
.
# 5. Math Libraries
.
.
# 6. Cougar YOD/FYOD
.
.
# 7. NQS
.<repeated until all tests are checked out of CVS>
.
.
.
echo ""
echo " >>>>>>  End of CVS CheckOut of 2-week Eval files <<<<<< "
echo ""
echo " Ending date/time: `date`"
echo ""
echo ""
```

<End of Script 1>

Script 2: status_checker

```
#!/bin/sh
echo " "
echo "       ********** STATUS CHECKER ********** "
echo "  file: /home/sigeval/bin/status_checker "
echo "     date: `date` "
echo " "

echo " >>> Command: getmagic "
getmagic
echo " "
echo " "

echo " >>> Command: showmesh "
showmesh
echo " "
echo " "

echo " >>> Command: /usr/local/etc/sw_version (inline version); cicc -V; f77 -V; f90 -V "
echo "Obtaining bootmagic information"
KERNEL_NAME=`/sbin/getmagic BOOT_KERNEL_NAME`
SERVER_NAME=`/sbin/getmagic BOOT_STARTUP_NAME`
EMULATOR_NAME=`/sbin/getmagic BOOT_EMULATOR_NAME`
PUMA_NAME=`/sbin/getmagic BOOT_ALT_KERNEL_NAME`
PCT_NAME="/cougar/sys/pct"
PGCC_COMPILER="/usr/pgi/osf86/bin/pgcc"
#
# Generate OS version information
#
echo "Generating OS version information"
echo "=----------------------------------------------------------------="
echo "                  Operating System Version Information"
echo ""
temp=`/usr/bin/strings /mach_servers/startup | grep 'TFLOPS O/S Release'| sed -e
'1s;^Paragon OSF;OSF;'`
version=`echo $temp | awk -F";" '{print $1}'`
built=`echo $temp | awk -F";" '{print $2}'`
echo " $version"
echo " Built on: $built"
echo " "
echo " Compiler version:    " `strings $PGCC_COMPILER |grep Rel | head -1`
echo " "
echo " OSF Kernel:  " `what $KERNEL_NAME | grep mach_kernel`
echo " OSF Server:  " `what $SERVER_NAME | grep vmunix`
echo " OSF Emulator:" `what $EMULATOR_NAME | grep emulator`
echo " Cougar QK:   " `what -s $PUMA_NAME`
echo " Cougar PCT:  " `what -s $PCT_NAME`
echo +----------------------------------------------------------------+
echo " "
cicc -V; f77 -V; f90 -V
echo " "
echo " "

echo " >>> Command: env "
env
```

```
echo " "
echo " "

echo " >>> Command: qstat -bl "
qstat -bl
echo " "
echo " "

echo " >>> Command: ps -elf"
ps -elf
echo " "
echo " "

echo " >>> Command: showfs -k"
showfs -k
echo " "
echo " "

echo "     ********** END OF STATUS CHECKER ********** "
echo "     date: `date` "
echo " "
echo " "
```

< end of Script 2 >

Script 3: Pinging Interactive Partition Until Specified Number of Nodes Becomes Available

```
# Define number of max nodes allowed
#  Janus' interactive partition always has 140 nodes allocated.
#  May use the following example on the command line to change default:
#    MAX_NODES=32; export MAX_NODES  (bourne shell)
#  or
#    setenv MAX_NODES 32 (c shell)
#

   MAX_NODES=${MAX_NODES:-140}
    if [ "$MAX_NODES" -gt 140 ] ; then
       MAX_NODES=140
    fi

# Define number of min nodes allowed
#  This value should not be changed unless absolutely required

   MIN_NODES=${MIN_NODES:-3}
    if [ "$MIN_NODES" -lt 3 ] ; then
       MIN_NODES=3
    fi

# Define number of tries to check available interactive nodes
number_of_tries=5
# Define number of tries remaining (initially = number_of_tries)
number_of_tries_left=$number_of_tries
# Define time between tries
number_of_seconds=10
# Define reduction factor (percent) to reduce number of nodes used
  REDUCTION_FACTOR=${REDUCTION_FACTOR:-10}
if [ "$REDUCTION_FACTOR" -lt 0 -o "$REDUCTION_FACTOR" -gt 90 ] ; then
    echo " "
    echo " ERROR in eat.auto script: REDUCTION_FACTOR "
    echo "  0 <= REDUCTION_FACTOR <= 90 percent "
    echo "  REDUCTION_FACTOR = $REDUCTION_FACTOR"
    echo " REDUCTION_FACTOR is out of bounds -- check eat.auto script "
    echo "  Script is exiting "
    echo " "
    exit 2
fi

echo " "
echo " number of attempts to determine interactive nodes = $number_of_tries"
echo " number of tries left = $number_of_tries_left "
echo " time between attempts = $number_of_seconds sec"
echo " reduction factor = $REDUCTION_FACTOR percent "
echo " maximum nodes allowed = $MAX_NODES "
echo " minimum nodes allowed = $MIN_NODES "
echo " "
echo " "
if [ "$MAX_NODES" -le "$MIN_NODES" ] ; then
   echo " "
```

```
    echo " ERROR: Max nodes allowed is less than Min nodes allowed."
    echo "  This needs to be corrected by the user."
    echo " "
    echo " Script exiting."
    echo " "
    exit 2
fi

# pre-define the number of interactive nodes
interactive_nodes1=0
interactive_nodes2=-1


while [ "$interactive_nodes1" -ne "$interactive_nodes2" -a  "$number_of_tries_left" -gt 0 ]

do

interactive_nodes1=`showmesh | grep interactive | awk '{ print $4}' `
echo " "

if [ "$interactive_nodes1" -gt "$MAX_NODES" ] ; then
    interactive_nodes1=$MAX_NODES
    REDUCTION_FACTOR=0
    echo " "
    echo " Since the Max number of allowed nodes is less than"
    echo "   the available number, no reduction factor is used."
    echo " "
fi

echo "Number of interactive nodes (1st check) = $interactive_nodes1"
if [ "$interactive_nodes1" -le "$MIN_NODES" ] ; then
    echo " "
    echo " ERROR: Not enough nodes to run EATs "
    echo "   Number of interactive nodes: $interactive_nodes1 "
    echo " "
    number_of_tries_left=`expr $number_of_tries_left - 1`
    echo "   Will try $number_of_tries_left more time(s)"
    echo " "
    sleep $number_of_seconds
    continue
else

# pause, then check number of available interactive nodes again, to
#  check that a load from another user was not in progress since the
#  first check
sleep $number_of_seconds
interactive_nodes2=`showmesh | grep interactive | awk '{ print $4}' `

if [ "$interactive_nodes2" -gt "$MAX_NODES" ] ; then
    interactive_nodes2=$MAX_NODES
fi

echo "Number of interactive nodes (2nd check) = $interactive_nodes2"
echo " "

if [ "$interactive_nodes1" -ne "$interactive_nodes2" ] ; then
```

```
    echo " "
    echo " ERROR: Number of available interactive nodes is changing, "
    echo "   probably due to heavy usage."
    echo " "
    number_of_tries_left=` expr $number_of_tries_left - 1 `
    echo "   Will try $number_of_tries_left more time(s)"
    echo " "
    sleep $number_of_seconds
    continue
fi

break

fi

done


    if [ "$number_of_tries_left" -le 0 ] ; then
        total_time_for_tries=` expr $number_of_tries \* $number_of_seconds `
        echo " "
        echo " Number of attempts made: $number_of_tries "
        echo "  Time between attempts: $number_of_seconds sec"
        echo "  Total time trying to allocate nodes: $total_time_for_tries sec"
        echo " "
        echo " Number of attempts has been exceeded."
        echo "    Wait awhile and try again later."
        echo " "
        echo " >>>  Exiting mini-EATs script"
        echo " "
        echo " "
        exit 100
   fi

in2=$interactive_nodes2
interactive_nodes=` expr $in2 - \( $REDUCTION_FACTOR \* $in2 \) \/ 100 `
nodes_available=$in2
nodes_after_reduction=$interactive_nodes

# Ensure nodes used are less than MAX_NODES, particulary if
#   MAX_NODES has been redefined by command line instruction (see above)
#     if [ "$interactive_nodes" -gt "$MAX_NODES" ] ; then
#         interactive_nodes=$MAX_NODES
#     fi

echo " "
echo " Max nodes allowed: $MAX_NODES"
echo " Min nodes allowed: $MIN_NODES"
echo " Nodes available to be used: $nodes_available"
nar=$nodes_after_reduction
echo " Nodes available after $REDUCTION_FACTOR percent reduction: $nar"
echo " "
echo " Number of interactive nodes to be used: $interactive_nodes "
echo " "

if [ "$interactive_nodes" -le "$MIN_NODES" ] ; then
```

```
    echo " "
    echo " ERROR: Not enough nodes to run EATs "
    echo "   Number of interactive nodes: $interactive_nodes "
    echo " "
    echo " "
    echo " Script is exiting"
    exit 100
fi

#=================================================================
========

# for virtual nodes (proc mode 3)
if [ "$COP_OPT" ]
then
      if [ `echo "$COP_OPT" | awk '{ print $2 }'` = "3" ]
      then

            echo " "
            echo " >>> COP_OPT = 3: proc 3 mode is set"
            echo " "

             NNODES_ALT=`/sbin/getmagic -w BOOT_ALT_NODE_LIST | wc -w | sed 's/
//g'`
             if [ "$NNODES_ALT" -gt 0 ] ; then
#                  NNODES=$NNODES_ALT
                  NNODES=`expr $NNODES \* 2`
                  SIZE=$NNODES
             fi
      fi
fi


# set YODPARAMS
# for virtual nodes (proc mode 3)
if [ "$COP_OPT" ]
then
      if [ `echo "$COP_OPT" | awk '{ print $2 }'` = "3" ]
      then
            YODPARAMS="-p 3"
      fi
else
      YODPARAMS=" "
fi

NNODES=$interactive_nodes
SIZE=$NNODES

export YODPARAMS
export SIZE
export NNODES
```

< end of Script 3 >

Appendix H: Sample Page from Test Log

TEST MATRIX, 90% CONFIDENCE LEVEL, 2-WEEK EVAL
OS VERSION: _____
EVAL'er(s): _____ DATES: _____

No.	TEST SUITE	TARGET SYSTEM	RUN TIME (APPROX HRS)	90% CONF TEST	EVAL NOTES and COMMENTS
1.	EATs for Basil	Basil (test requires nqs and raids) Logfile directory: **/home/sigeval/ Accept_x86_R4_3_5** (for example)	see Note 1, below, for Basil statistics	X	PASS/FAIL (P/F): START: END: COMMENTS: Core1: Tools: Xdebug: Core2: CougarP3: Signal: CougarP0:
2.	mini-EATs for Janus	Janus (test does NOT require nqs nor raids; is a modified subset of original tests) Logfile directory:	See Note 1, below, for Janus stats (pending)		PASS/FAIL (P/F): START: END: COMMENTS: Typically, this test is not run
3.	Parallel Apps / Message-Passing	Basil; Can be run on Rosemary, but the test message/nx/gsend_gsum will fail to run; re-run this test on Basil Logfile directory: **$T92DIR/Auto**	16	X	PASS/FAIL (P/F): START: END: COMMENTS:
4.	MPI	Basil Logfile directory: **$T92DIR/MPITEST/ Test**	28	X	PASS/FAIL (P/F): START: END: COMMENTS:
5.	Math Libraries	Basil or Rosemary Logfile directory: **$T92DIR/lib_tests**	26 hrs for rosemary	X	PASS/FAIL (P/F): START: END: COMMENTS:

Appendix I: Test Plan for R4.5.2 (full build)

********** Test Plan for 4.5.2**********

Janus Eval Test Date: Thursday, January 20, 2005
Janus Install Target Date: Thursday, February 17, 2005

PSE on duty:	Sean Taylor	srtayl@sandia.gov <phone #'s have been deleted)
Eval'er:	Daniel Barnette	dwbarne@sandia.gov
Other contacts relevant to this eval:		
	Bob Benner	rebenne@sandia.gov
	John VanDyke	jpvandy@gabe.sandia.gov

GOALS:

I) Validate R4.5.2 "full build" by running MUNOPS on Janus

This build address the following PR's:

BobB	926	Memory fragmentation issue
BobB	1408	Reduce malloc header from 24 to 16 bytes
BobB	1450	libpuma and utlib/yod version numbers outdated.
Paul	1502	Default umask file creation setting
BobB	1545	Fix mpi2c++ include path for mpich v. 1.2.4
BobB	1546	Update lapack library to v. 3.0
BobB	1588	Remove diagnostic output from malloc failure
BobB	1598	Make libdbmalloc link-compatible with new puma malloc
BobB	1600	Fix MPI_SHORT_MSG_SIZE handling in MPICH v. 1.2.4

Eval Preparations/Tasks for SWE's:

1. Make sure MUNOPS is on Janus in /home/projects/eval/munops/R4_5_2_munops_split. (Daniel)

2. Create directory for logfiles (Daniel)
   a. mkdir /home/projects/eval/Jan20_2005_logfiles
   b. chmod 775 /home/projects/eval/Jan20_2005_logfiles

Eval Preparations/Tasks for PSE's:

1. Create R4.5.2 boot disk for Janus (PSE)

2. On the morning of 01/20/05, send email to janus-users@sandia.gov with the following text (PSE):

==========
Janus (unclassified) will be unavailable TODAY, Thursday, 01/20/05, all day, from 6:00 a.m. until 11:00 p.m. We will be performing an operating system evaluation, preventive maintenance, and a color change.

If you have any questions and/or concerns, send email to janus-help@sandia.gov.
==========

Eval Goal I

1. At approximately 6:00AM start system shutdown on janus from console giving users 10 minutes (PSE):

```
shutdown -h +10
```

2. Put R4.5.2 disk in place and boot janus-eval (PSE)

3. Prevent users from accessing the system (PSE)
   -- Keep /etc/nologin in place

4. Do not start scheduling or queues! (PSE)

5. Change root password on janus. (PSE)

6. Since MUNOPS (and not IO-MUNOPS) is being run, umount all pfs directories, and all ufs directories except /home/projects (PSE):

```
cd /home/projects          # to keep eval disk busy & mounted
umount –a –t pfs
umount –s –t ufs
```

7. Make sure /, /home/projects, and /tmp are still mounted (PSE)
   df

8. Contact janus-sw via email and advise:
   "System is ready for Goal I." (PSE)

9. Run portion of MUNOPS that requires root privilege. (Daniel)

```
cd /home/projects/eval/munops/R4_5_2
status_checker > logfile_status_checker_begin.janus_dwbarne_012005
chown dwbarne status_checker_begin.janus_dwbarne_jan20_2005
make_parts_tfmunops.ksh        #also stops nqs
clean_tmp                      # cleans /scratch directory of previous files
```

10. Start MUNOPS (Daniel):

```
run_munops.ksh > logfile_munops_split.janus_dwbarne_012005
```

11. Monitor MUNOPS to ensure it is making progress. (Daniel)

12. In case of system failure/hang (PSE):
   a) collect stampede;
   b) reboot;
   c) create case;
   d) Return to Step 10

Repeat as necessary.

13. At 3:00PM:
   a) make tentative determination if R4.5.2 is a go or no-go;
   b) send email of current status to Bob, John, PSE's. (Daniel)

Goal I pass criteria:

a) MUNOPS runs without crashing the system; OR

b) If system crashes, the crash can be attributed to a known problem.

14. At 5:00PM, as root, run (Daniel):

```
cd /home/projects/eval/munops/R4_5_2
delete_parts_tfmunops.ksh          #hot-stop of MUNOPS
status_checker > status_checker_end.janus_dwbarne_jan20_2005
chown dwbarne status_checker_end.janus_dwbarne_jan20_2005
```

15. Copy system files off R4.5.2 boot disk. (Daniel)

```
cd /home/projects/eval/Jan2005_logfiles
cp /var/adm/compute/run.log .
cp /var/adm/*.log .
cp /var/adm/syslog/*.log .
chmod 664 *
chown dwbarne:wg-intel *
```

17. Send email to janus-sys and janus-sw announcing: (Daniel)

a) the end of eval;

b) add a statement as to how many times MUNOPS had to be started;

c) state whether R4.5.2 is a go or no-go

18. Shutdown R4.5.2 and revert to current-production R4.4.4 boot disk. (PSE)

Goal I pass criteria: munops continues to run without crashing the system.

----------END----------

Appendix J: Script for Gathering Test Results

```
#!/bin/csh
#
# Filename: gather_results
#
# To run:
# First, modify User-defined directory names within file; then
# Execution command: utility_untar_2wk_evals > logfile_untar_2wk_evals &
# Monitor logfile for progress and completion
#
# Author: D. W. Barnette
# Date: July, 2002
#

if ( $#argv == 1 ) goto goodargs
echo " "
echo " USAGE ERROR: You must type the name of the OS version (ex: R4_2_2)"
echo "              on the command line."
echo "   Usage: utility_untar_2wk_evals OS_VERSION"
echo " "
exit 1

goodargs:

echo " "
echo "   *****  GATHER RESULTS FILES ***** "
echo ""
echo " Starting date/time: `date`"

setenv OS_VERSION $1
setenv USER
echo " "
echo " OS_VERSION = $1 "
echo " "

#1. Eats
setenv EVALDIR1   /Net/intel-swe/sigeval/Accept_x86_"$OS_VERSION"

#2. mini-Eats
setenv EVALDIR2   /Net/usr/home/$LOGNAME/sigeval/Accept_x86_"$OS_VERSION"

#3. Parallel Apps
setenv EVALDIR3   /Net/intel-swe/sigeval/"$OS_VERSION"/Auto

#4. MPI
setenv EVALDIR4   /Net/intel-swe/sigeval/"$OS_VERSION"/MPITEST/Test

#5. Math Libs
setenv EVALDIR5   /Net/intel-swe/sigeval/"$OS_VERSION"/lib_tests

#6. Cougar Yod/Fyod
setenv EVALDIR6   /Net/intel-swe/sigeval/"$OS_VERSION"/Auto/cougar

#7. NQS
setenv EVALDIR7   /Net/intel-swe/sigeval/"$OS_VERSION"/TF_NQS

#8. General Regression
setenv EVALDIR8   /Net/intel-swe/sigeval/"$OS_VERSION"/regress

#9. MUNOPS
setenv EVALDIR9a  /Net/intel-swe/projects/eval/munops/"$OS_VERSION"
```

```
setenv EVALDIR9b  /Net/intel-swe/projects/eval/io_munops/"$OS_VERSION"

#10. DDT File I/O
setenv EVALDIR10a /Net/intel-swe/sigeval/"$OS_VERSION"
setenv EVALDIR10b /Net/intel-swe/sigeval/"$OS_VERSION"/AutoRoot/cmd/pfs

#11. Other/Non-DDT File I/O
setenv EVALDIR11  /Net/intel-swe/sigeval/"$OS_VERSION"/io

#12. AutoUnix
setenv EVALDIR12  /Net/intel-swe/sigeval/"$OS_VERSION"/AutoUnix

#13. DDT Sockets
setenv EVALDIR13  /Net/intel-swe/sigeval/"$OS_VERSION"/Auto/sockets

#14. AutoDebug
setenv EVALDIR14  /Net/intel-swe/sigeval/"$OS_VERSION"/Autodebug/cases

#15. AutoTprof
setenv EVALDIR15  /Net/intel-swe/sigeval/"$OS_VERSION"/AutoTprof


#create directory for storing all results
setenv STORE_SUB_DIR RESULTS_FILES_`date +"%T%h%d%Y%a"`
#setenv STORE_DIR /Net/usr/home/$LOGNAME/RESULTS_FILES_`date +"%T%h%d%Y%a"`
setenv STORE_DIR /Net/usr/home/$LOGNAME/"$STORE_SUB_DIR"
mkdir $STORE_DIR
echo ""
echo " STORAGE DIRECTORY: $STORE_DIR"
echo ""


#1. EATS
echo ""
echo "================================================"
echo " 1. EATS"
# make the results directory
mkdir $STORE_DIR/EATS

#if eval directory exists, get results files
echo ""
echo " Searching for results directory $EVALDIR1"
if (! -x "$EVALDIR1") then
        echo ""
        echo " >> Directory not found: $EVALDIR1"
        touch $STORE_DIR/EATS/no_results
else
        echo ""
        echo " >> Directory found: $EVALDIR1"
        cp $EVALDIR1/WW*.results.* $STORE_DIR/EATS
setenv FILE_COUNT 0
setenv FILE_COUNT ` ls -1 $STORE_DIR/EATS/ | grep -lc '^[a-zA-Z]' `

if ( $FILE_COUNT == 0 ) then
                echo "   No results found to copy to directory"
                touch $STORE_DIR/EATS/no_results
        else
                echo "   $FILE_COUNT results files written to $STORE_DIR/EATS"
        endif
endif


#2. mini-EATS
```

```
echo ""
echo "================================================"
echo " 2. MINI-EATS"
# make the results directory
mkdir $STORE_DIR/MINI_EATS

#if eval directory exists, get results files
echo ""
echo " Searching for results directory $EVALDIR2"
if (! -x "$EVALDIR2") then
        echo ""
        echo " >> Directory not found: $EVALDIR2"
        touch $STORE_DIR/MINI_EATS/no_results
else
        echo ""
        echo " >> Directory found: $EVALDIR2"
        cp $EVALDIR1/WW*.results.* $STORE_DIR/MINI_EATS
setenv FILE_COUNT 0
setenv FILE_COUNT ` ls -1 $STORE_DIR/MINI_EATS/ | grep -lc '^[a-zA-Z]' `

if ( $FILE_COUNT == 0 ) then
                echo "   No results found to copy to directory"
                touch $STORE_DIR/MINI_EATS/no_results
        else
                echo "   $FILE_COUNT results files written to
$STORE_DIR/MINI_EATS"
        endif
endif


#3. Parallel Apps
echo ""
echo "================================================"
echo " 3. PARALLEL_APPS"
# make the results directory
mkdir $STORE_DIR/PARALLEL_APPS

#if eval directory exists, get results files
echo ""
echo " Searching for results directory $EVALDIR3"
if (! -x "$EVALDIR3") then
        echo ""
        echo " >> Directory not found: $EVALDIR3"
        touch $STORE_DIR/PARALLEL_APPS/no_results
else
        echo ""
        echo " >> Directory found: $EVALDIR3"
        cp $EVALDIR1/WW*.results.* $STORE_DIR/PARALLEL_APPS
setenv FILE_COUNT 0
setenv FILE_COUNT ` ls -1 $STORE_DIR/PARALLEL_APPS/ | grep -lc '^[a-zA-Z]' `

if ( $FILE_COUNT == 0 ) then
                echo "   No results found to copy to directory"
                touch $STORE_DIR/PARALLEL_APPS/no_results
        else
                echo "   $FILE_COUNT results files written to
$STORE_DIR/PARALLEL_APPS"
        endif
endif

.
.
< Coding omitted for brevity >
```

```
.
.


cd /Net/usr/home/$LOGNAME
echo ""
echo " Tar'ing the results directory ... please wait ..."
echo ""
tar -cvf $STORE_SUB_DIR.tar $STORE_SUB_DIR
echo ""
echo ""
echo "    Results tar file created: $STORE_SUB_DIR.tar"
echo "     in directory /Net/usr/home/$LOGNAME"
echo ""
echo "" Compressing the tar file ... please wait ..."
echo ""
compress $STORE_SUB_DIR.tar
echo "    Tar file compressed."
echo ""

echo ""
echo " >>>>>>  End of Gathering Results Files <<<<<< "
echo ""
echo " Ending date/time: `date`"
echo ""
```

Appendix K: *Mathematica* Script for Plotting Test Results

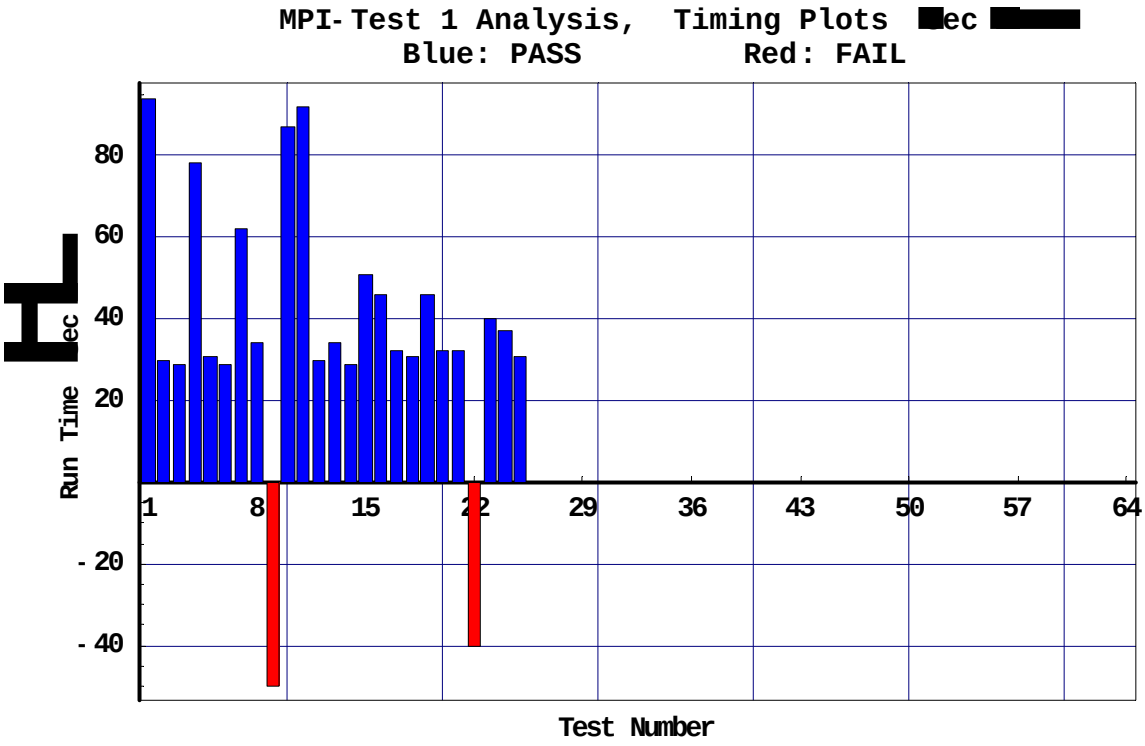
```
(*
  Filename: mpitest1_bar_graphs.nb
      Written Oct 2000, by D. Barnette
  *)
 Clear[
  LogFile,
  Description1, Description2,
  TodaysDateAndTime,
  RowsColumns,
  NumberOfTests,
  TimeOfStartHours,
  TimeOfStartMinutes,
  TimeOfStartSeconds,
  TimeOfFinishHours,
  TimeOfFinishMinutes,
  TimeOfFinishSeconds,
  DateOfTest,
  TimeOfTest,
  TestNumber,
  Test,
  DataWord
  ]

 (* Search logfile for number of tests run by searching
    for the word "Test:". For each successful find, increment
NumberOfTests and reset DataWord to "NULL" and start again, until
"EndOfFile" is detected *)

NumberOfTests=0;
DataWord="NULL";
LogFile=OpenRead["d:\Intel_MPITest\logfile_incomplete.txt"];While[DataWo
rd≠"EndOfFile",
  While[(DataWord≠"Test:") && (DataWord≠"EndOfFile"),
    DataWord=Read[LogFile,Word];
    ];
  If[DataWord=="Test:",
    NumberOfTests+=1;
    (* Always reset DataWord before beginning next search *)
    DataWord="NULL";
    ];
    ];
Close[LogFile]
 d:\Intel_MPITest\logfile_incomplete.txt
 NumberOfTests
 25
 TimeOfTest=Table[0,{NumberOfTests}];
 Dimensions[TimeOfTest]
 {25}
 Test=Table[0,{NumberOfTests}];
 Dimensions[Test]
 {25}
 TestNumber=Table[0,{NumberOfTests}];
 Dimensions[TestNumber]
 {25}
 Date[]
 {2001,10,12,19,32,48}
 Description1="MPI-Test 1 Analysis";
 Description2=" Timing Plots (sec)";
 TodaysDateAndTime:=(
    Temp=Date[];
    StringForm[
```

```
      "``   Date: ``/``/``    Time: ``:``:``",
      Description1<>", "<>Description2,
      Temp[[2]],Temp[[3]],Temp[[1]],
      Temp[[4]],Temp[[5]],Temp[[6]]
      ]
    )
 TDT=TodaysDateAndTime
 MPI-Test 1 Analysis,  Timing Plots (sec)   Date: 10/12/2001    Time:
19:32:48
 (* Data from 'logfile...' *)
 Column1="Date";
Column2="Duration (sec)";
Column3="Eval Computer";
Column4="Test";
 (* Uncomment following to check if file can be opened; for debugging
code *)
  (* !!"d:\Program
Files\DevStudio\MyProjects\Robug_Simulator\RobocopOutputMod0.txt" *)
 LogFile=OpenRead["d:/Intel_MPITest/logfile_incomplete.txt"];
 (* Skip first 4 lines *)
 For[k=1,k≤4,k++,
  Skip[LogFile, Record]
  ]
 For[k=1,k≤NumberOfTests,k++,

    (* Sequentially number tests *)
    TestNumber[[k]]=k;

    DateOfTest=Read[LogFile,Word];

    TimeOfStartHours=Read[LogFile,Number];
    Skip[LogFile,Character];
    TimeOfStartMinutes=Read[LogFile,Number];
    Skip[LogFile,Character];
    TimeOfStartSeconds=Read[LogFile,Number];

    ComputerName=Read[LogFile,Word];

    Skip[LogFile, Word];

    Test[[k]]=Read[LogFile,Word];

    Skip[LogFile,Record];
    Skip[LogFile,Record];
    Skip[LogFile,Word];

    TimeOfFinishHours=Read[LogFile,Number];
    Skip[LogFile,Character];
    TimeOfFinishMinutes=Read[LogFile,Number];
    Skip[LogFile,Character];
    TimeOfFinishSeconds=Read[LogFile,Number];

    Skip[LogFile,Record];

    If[TimeOfFinishHours>=TimeOfStartHours,
      TimeOfTest[[k]]=(TimeOfFinishHours-TimeOfStartHours)*3600+
          (TimeOfFinishMinutes-TimeOfStartMinutes)*60+
          (TimeOfFinishSeconds-TimeOfStartSeconds)
      ,
      TimeOfTest[[k]]=(TimeOfFinishHours+24-TimeOfStartHours)*3600+
          (TimeOfFinishMinutes-TimeOfStartMinutes)*60+
          (TimeOfFinishSeconds-TimeOfStartSeconds)
      ];
    ];
```

```
Close[LogFile]
 d:/Intel_MPITest/logfile_incomplete.txt
 DateOfTest
 10/01/01
 ComputerName
 basil
 Dimensions[TimeOfTest]
 {25}
 TimeOfTest

{94,30,29,78,31,29,62,34,50,87,92,30,34,29,51,46,32,31,46,32,32,40,40,37
,31}
 Dimensions[Test]
 {25}


 (* Get Pass/Fail information from 'results...' file generated by MPI
eval test, and match with info from 'logfile..' above *)

 PFfile=OpenRead["d:/Intel_MPITest/results.cgr1_18.txt"];

(* Skip 14 lines before reading *)
Do[
    Skip[PFfile,Record],
    {j,1,12}
    ];

Do[
    PassFail=Read[PFfile,Word];

    If[PassFail≠"EndOfFile",

      While[
        TestPF≠"M",
        TestPF=Read[PFfile,Character]
        ];

      TestPF=Read[PFfile,Word];
      TestPF="M"<>TestPF;
      Print[j,"  ",PassFail,"  ",TestPF,"  ",Test[[j]]];

      If[TestPF□Test[[j]],
        If[PassFail≠"P",
          TimeOfTest[[j]]=-TimeOfTest[[j]];
          Print["     >>> NOTE: TimeOfTest[[",j,"]] changed to negative
for plotting  "];
          Print[" "];
          ],
        Print[" Error: For j = ",j,"/",NumberOfTests,",  TestPF =
",TestPF,",  Test[[j]] = ",Test[[j]],"  " ];
        Print[" "];
        ],
      Print[" "];
      Print[" EndOfFile read for ",PFfile,".   j = ",j,"   NumberOfTests
= ",NumberOfTests];
      Print[" "];

      ],
    {j,1,NumberOfTests}
    ];
Close[PFfile]
 1    P    Mc/blocking/functional/MPI_Bsend_ator    MPI_Bsend_ator
```

```
  Error: For j =  1 / 25 ,  TestPF =
Mc/blocking/functional/MPI_Bsend_ator ,  Test[[j]] =  MPI_Bsend_ator

 2    P    MPI_Bsend_null    MPI_Bsend_null
 3    P    MPI_Bsend_overtake    MPI_Bsend_overtake
 4    P    MPI_Bsend_rtoa    MPI_Bsend_rtoa
 5    P    MPI_Recv_comm    MPI_Recv_comm
 6    P    MPI_Recv_null    MPI_Recv_null
 7    P    MPI_Recv_pack    MPI_Recv_pack
 8    P    MPI_Rsend_null    MPI_Rsend_null
 9    F    MPI_Rsend_rtoa    MPI_Rsend_rtoa
      >>> NOTE: TimeOfTest[[ 9 ]] changed to negative for plotting

 10    P    MPI_Send_ator    MPI_Send_ator
 11    P    MPI_Send_ator2    MPI_Send_ator2
 12    P    MPI_Send_null    MPI_Send_null
 13    P    MPI_Send_off    MPI_Send_off
 14    P    MPI_Send_overtake    MPI_Send_overtake
 15    P    MPI_Send_rtoa    MPI_Send_rtoa
 16    P    MPI_Ssend_ator    MPI_Ssend_ator
 17    P    MPI_Ssend_null    MPI_Ssend_null
 18    P    MPI_Ssend_overtake    MPI_Ssend_overtake
 19    P    MPI_Ssend_rtoa    MPI_Ssend_rtoa
 20    P    MPI_Allgather    MPI_Allgather
 21    P    MPI_Allgatherv    MPI_Allgatherv
 22    F    MPI_Allreduce    MPI_Allreduce
      >>> NOTE: TimeOfTest[[ 22 ]] changed to negative for plotting

 23    P    MPI_Allreduce_loc    MPI_Allreduce_loc
 24    P    MPI_Allreduce_user    MPI_Allreduce_user
 25    P    MPI_Alltoall    MPI_Alltoall
 d:/Intel_MPITest/results.cgr1_18.txt


 (*
  Get Graphics packages needed for plots
  *)
 <<Graphics`Graphics`
 <<Graphics`Legend`
 units[xmin_,xmax_]:=Range[Floor[xmin],Floor[xmax],7]
 (* Tests to be plotted with bar charts in increments of 64 tests *)
incr=64;
 TickMarkList=units[1,incr]
 {1,8,15,22,29,36,43,50,57,64}
 LengthOfList=Length[TickMarkList]
 10
 Clear[TickMarkArray]
 Array[TickMarkArray,{LengthOfList,2}];
 TickMarkArray=Table[0,{LengthOfList},{2}]
 {{0,0},{0,0},{0,0},{0,0},{0,0},{0,0},{0,0},{0,0},{0,0},{0,0}}
 Table[Dimensions[TickMarkArray]]
 {10,2}
 (* TableForm[TickMarkArray] *)
 jjmod=Mod[NumberOfTests,incr]
 25
 If[jjmod==0,
  jjappend=0,
  jjappend=incr-jjmod;
  Clear[TestAdd, TimeAdd];
  Array[TestAdd,{jjappend},{1}];
  TestAdd=Table[0,{jjappend},{1}];
  TimeAdd=Table[0,{jjappend},{1}];
  NumberOfBars=NumberOfTests+jjappend;
  Do[
```

```
    TestAdd[[jj]]= " ",
    {jj,1,jjappend}
    ];
  Do[
    TimeAdd[[jj]]=0.0,
    {jj,1,jjappend}
    ];
  Test = Join[Test,TestAdd];
  TimeOfTest=Join[TimeOfTest,TimeAdd];
  ]
 NumberOfTests
 25
 NumberOfBars
 64
 Dimensions[Test]
 {64}
 jjappend
 39
 jjmax=NumberOfBars
 64
 Do[
  Do[
    TickMarkArray[[jtick,1]]=TickMarkList[[jtick]];
    TickMarkArray[[jtick,2]]=ToString[TickMarkList[[jtick]]+jj-1],
    {jtick,1,LengthOfList}
    ];

  BarChart[
    Table[TimeOfTest[[kk]],{kk,jj,jj+ incr-1} ],
    ImageSize → 600,
    Ticks→{TickMarkArray,Automatic},
      PlotLabel→StyleForm[ Description1<> ", "<> Description2  <> "\n
Blue: PASS         Red: FAIL",  FontSize→15, FontWeight→Bold],
     TextStyle→{FontSize→12,FontWeight→Bold},
     BarOrientation→Vertical,
     Axes→True,
    AxesStyle→Thickness[0.0040],
     Frame→True ,
    FrameTicks→None,
    FrameLabel->{"Test Number","Run Time (sec) \n  "},
RotateLabel→True,
     BarStyle→(Which[#>0,RGBColor[0,0,1],#<0,RGBColor[1,0,0]]&),
    GridLines→Automatic,
    RotateLabel->True
    ] ,
  {jj,1,jjmax,incr}
  ]
```




```
(* Output all tests in tabular format with 3 columns max *)

ColumnsMax=3;
RowsList=Floor[NumberOfBars/(ColumnsMax-1)];
Extras=Mod[NumberOfBars,ColumnsMax-1];
If[Extras□0,
    Table0=Table[
        {"--","---------------","--","---------------"},
        {j,1,1}
        ],
    Table0=Table[
        {"--","---------------","--","---------------","--","-----------
----"},
        {j,1,1}
        ]
    ];
Table1=Table[
      {
        j, Test[[j]], j+RowsList,Test[[j+RowsList]],
        j+2*RowsList,Test[[j+2*RowsList]]},
      {j,1,Extras}
      ];
Table2=Table[
      {
        j, Test[[j]], j+RowsList,Test[[j+RowsList]]},
      {j,Extras+1,RowsList}
      ];
TestTable=Join[Table0,Table1,Table2];
If[Extras□0,
  TableForm[
    TestTable, TableHeadings→{None,{"No." , "Test","No.","Test"}}
    ],
  TableForm[
    TestTable, TableHeadings→{None,{"No." ,
"Test","No.","Test","No.","Test"}}
```

```
    ]
  ]
```

No.	Test	No.	Test
1	MPI_Bsend_ator	33	
2	MPI_Bsend_null	34	
3	MPI_Bsend_overtake	35	
4	MPI_Bsend_rtoa	36	
5	MPI_Recv_comm	37	
6	MPI_Recv_null	38	
7	MPI_Recv_pack	39	
8	MPI_Rsend_null	40	
9	MPI_Rsend_rtoa	41	
10	MPI_Send_ator	42	
11	MPI_Send_ator2	43	
12	MPI_Send_null	44	
13	MPI_Send_off	45	
14	MPI_Send_overtake	46	
15	MPI_Send_rtoa	47	
16	MPI_Ssend_ator	48	
17	MPI_Ssend_null	49	
18	MPI_Ssend_overtake	50	
19	MPI_Ssend_rtoa	51	
20	MPI_Allgather	52	
21	MPI_Allgatherv	53	
22	MPI_Allreduce	54	
23	MPI_Allreduce_loc	55	
24	MPI_Allreduce_user	56	
25	MPI_Alltoall	57	
26		58	
27		59	
28		60	
29		61	
30		62	
31		63	
32		64	

Internal Distribution:

MS	Org	Name	Copies
0321	1400	W. Camp	1
0370	1411	S. Mitchell	1
0310	1412	M. Rintoul	1
1110	1414	S. Collis	1
1110	1415	S. Rountree	1
1111	1416	A. Salinger	1
0316	1420	S. Dosanjh	1
0316	1420	J. Tomkins	1
0376	1421	T. Blacker	1
0817	1422	J. Ang	1
0817	1422	D. Doerfler	1
0817	1422	S. Goudy	1
0817	1422	J. VanDyke	1
0817	1422	D. Barnette	5
0817	1422	R. Benner	1
0817	1422	S. Kelly	1
0817	1422	J. Stearley	1
1110	1423	N. Pundit	1
0822	1424	D. White	1
0321	1430	J. Nelson	1
0378	1431	R. Summers	1
0378	1433	J. Strickland	1
1110	1435	J. Aidun	1
0316	1437	S. Hutchinson	1
0813	4311	M. Cahoon	1
0823	4324	C. Leishman	1
0822	4326	D. Pavlakos	1
0807	4328	J. Noe	1
0807	4328	M. Davis	1
0807	4328	R. Ballance	1
0807	4328	M. Barnaby	1
0807	4328	F. Jaramillo	1
0807	4328	V. Kuhns	1
0807	4328	P. Sanchez	1
0807	4328	S. Taylor	1
0805	4329	B. Swartz	1
0823	4320	J. Zepper	1
0832	4335	J. Dexter	1
0806	4336	L. Stans	1
9151	8900	J. Handrock	1
9158	8961	M. Sukalski	1
9152	8963	J. Friesen	1
0899	Technical Library		2
9018	Central Technical Files		2

External Distribution:

Shailendra Save 2
Cray Inc.
411 First Ave S.
Suite 600
Seattle, WA 98104